# Cross-Layer Fault Tolerance in Networks-on-Chip

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Gert Schley

aus Karlsruhe

| | |
|---|---|
| Hauptberichter: | Prof. Dr.-Ing. Martin Radetzki |
| Mitberichter: | Prof. Dr. Oliver Bringmann |
| | |
| Tag der mündlichen Prüfung: | 11. Juni 2018 |

Institut für Technische Informatik der Universität Stuttgart

2018

# Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Stuttgart, 01. Juni 2017

Gert Schley

*To my mom & dad and to my friends*

# Acknowledgements

This thesis would not have been possible without the support of many people over the years. I am so thankful to every single one of you.

I am deeply grateful for the unconditional support and love of my parents and for their trust in me and my work. I thank you for everything from the bottom of my heart.

I would like to thank Prof. Dr.-Ing. Martin Radetzki for giving me the opportunity to work at Embedded Systems department and for his honest and constructive feedback concerning my work. I also thank Prof. Dr. Oliver Bringmann for evaluating my thesis.

A big thank you to Laura for her friendship and her positive attitude that always motivated me to continue with my dissertation. ¡Muchas gracias! Another big thank you to Marcus for reading chapters of my thesis and for giving priceless feedback. ¡*Oiner geht noch*, my friend!

For being such great colleagues and friends I thank Weiyun, Adan, Rauf, Manuel, Leandro, Sabine, Mirjam, Helmut, Lothar, Eric, and Chang. When thinking of you I remember all the funny discussions and the absolutely great time we had.

Thank you to my dear friends Markus, Natalie V., Natalie R., Silvia, Katrin, Judith, Alex, Christoph and Inge for always being there for me. I am so glad to got to know all of you. CU.

For supporting me and my work I like to thank my former master students Dana, Ibrahim, Muhammad, and Nikolaos. You guys rock!

Speaking of Rock: last but not least I want to thank Blaze Bayley and Absolva for their great music and powerful gigs that helped me to take the *One More Step* to complete this work.

Stuttgart, June 2018
Gert Schley

# Contents

# List of Figures

# List of Tables

# Abbreviations

| | |
|---|---|
| **ACU** | Availability Communication Unit |
| **API** | Application Programming Interface |
| **BER** | Bit Error Rate |
| **CRM** | Communication Resource Management |
| **DFM** | Data Flow Management |
| **DOR** | Dimension-Order Routing |
| **ECC** | Error Correction Code |
| **EDC** | Error Detection Code |
| **ETG** | Enhanced Topology Graph |
| **FD** | Functional Diagnosis |
| **HNC** | Hierarchical Network Configuration |
| **IF** | Interface |
| **NI** | Network Interface |
| **NoC** | Network-on-Chip |
| **OSI** | Open Systems Interconnection Model |
| **P** | Diagnosis Protocol |
| **PE** | Processing Element |
| **RHR** | Reconfigurable Hierarchical Routing |
| **RCU** | Reconfiguration Unit |
| **SAT** | Satisfiability |
| **SBR** | Software-based Packet Rerouting |
| **SD** | Structural Diagnosis |
| **SER** | Soft Error Rate |
| **SW** | Switch |
| **TAM** | Test Access Mechanism |
| **VC** | Virtual Channel |

# Abstract

The design of Networks-on-Chip follows the Open Systems Interconnection (OSI) reference model. The OSI model defines strictly separated network abstraction layers and specifies their functionality. Each layer has layer-specific information about the network that can be exclusively accessed by the methods of the layer. Adhering to the strict layer boundaries, however, leads to methods of the individual layers working in isolation from each other. This lack of interaction between methods is disadvantageous for fault diagnosis and fault tolerance in Networks-on-Chip as it results in solutions that have a high effort in terms of the time and implementation costs required to deal with faults.

For Networks-on-Chip cross-layer design is considered as a promising method to remedy these shortcomings. It removes the strict layer boundaries by the exchange of information between layers. This interaction enables methods of different layers to cooperate, and thus, deal with faults more efficiently. Furthermore, providing lower layer information to the software allows hardware methods to be implemented as software tasks resulting in a reduction of the hardware complexity.

The goal of this dissertation is the investigation of cross-layer design for fault diagnosis and fault tolerance in Networks-on-Chip. For fault diagnosis a scheme is proposed that allows the interaction of protocol-based diagnosis of the transport layer with functional diagnosis of the network layer and structural diagnosis of the physical layer by exchanging diagnostic information. The techniques use this information for optimizing their own diagnosis process. For protocol-based diagnosis on the transport layer, a diagnosis protocol is proposed that is able to locate faulty links, switches, and crossbar connections. For this purpose, the technique utilizes available information of lower layers. As proof of concept for the proposed interaction scheme, the diagnosis protocol is combined with a functional and a structural diagnosis approach and the performance and diagnosis quality of the resulting combinations is investigated. The results show that the combinations of the diagnosis protocol with one of the lower layer techniques have a considerably reduced fault localization latency compared to the functional and the structural standalone techniques. This reduction, however, comes at the expense of a reduced diagnosis quality.

In terms of fault tolerance, the focus of this dissertation is on the design and implementation of cross-layer approaches utilizing software methods to provide fault tolerance for network layer routings. Two approaches for different routings are presented.

The requirements to provide information of lower layers to the software using the available Network-on-Chip resources and interfaces for data communication are discussed. The concepts of two mechanisms of the data link layer are presented for converting status information into communicable units and for preventing communication resources from being blocked. In the first approach, software-based packet rerouting is proposed. By incorporating information from different layers, this approach provides fault tolerance for deterministic network layer routings. As specialization of software-based rerouting, dimension-order XY rerouting is presented. In the second approach, a reconfigurable routing for Networks-on-Chip with logical hierarchy is proposed in which cross-layer interaction is used to enable hierarchical units to manage themselves autonomously and to reconfigure the routing. Both approaches are evaluated regarding their performance as well as their implementation costs.

In a final study, the cross-layer diagnosis technique and cross-layer fault tolerance approaches are combined. The information obtained by the diagnosis technique is used by the fault tolerance approaches for packet rerouting or for routing reconfiguration. The combinations are evaluated regarding their impact on Networks-on-Chip performance. The results show that the cross-layer information exchange with software has a considerable impact on performance when the amount of information becomes too large. In case of cross-layer diagnosis, however, the impact on Networks-on-Chip performance is significantly lower compared to functional and structural diagnosis.

# Zusammenfassung

Das Design von Networks-on-Chip orientiert sich am Open Systems Interconnection (OSI) Referenzmodell. Dieses definiert strikt voneinander abgegrenzte Netzwerkabstraktionsschichten und spezifiziert deren Funktionalität. Jede der Schichten besitzt Informationen über das Netzwerk, welche exklusiv durch die Methoden dieser Schicht genutzt werden können. Das Einhalten der strikten Schichtgrenzen bedingt jedoch, dass Methoden der einzelnen Schichten isoliert voneinander arbeiten. Dieser Interaktionsmangel zwischen den Methoden stellt einen Nachteil im Hinblick auf die Diagnose und die Tolerierung von Fehlern in einem Network-on-Chip dar und resultiert in Lösungen, welche einen erhöhten Aufwand bezüglich der Durchführungszeit sowie der Implementierungskosten für die Behandlung von Fehlern besitzen.

Schichtenübergreifendes Design dagegen wird im Bereich der Networks-on-Chip als ein erfolgversprechendes Mittel angesehen, um diesen Defiziten zu begegnen. Es hebt die strikten Schichtgrenzen durch den Austausch von Informationen zwischen den Schichten auf. Diese Interaktion ermöglicht Methoden unterschiedlicher Schichten, Fehler kooperativ zu behandeln und bietet dadurch die Möglichkeit, Fehler effizienter zu behandeln. Ferner ermöglicht es durch das Bereitstellen von Informationen niedrigerer Schichten an die Software, Hardware-Methoden als Software-Tasks zu implementieren und dadurch die Hardware-Komplexität von Networks-on-Chip zu reduzieren.

Das Ziel dieser Dissertation ist die Untersuchung von schichtenübergreifendem Design für Fehlerdiagnose und Fehlertoleranz in Networks-on-Chip. Für die Fehlerdiagnose wird ein Schema vorgeschlagen, welches die Interaktion von protokoll-basierter Diagnose der Transportschicht (transport layer) mit funktionaler und struktureller Diagnose der Vermittlungs- (network layer) beziehungsweise der Bitübertragungsschicht (physical layer) durch den Austausch von Diagnoseinformationen ermöglicht. Die Diagnosetechniken setzen dabei die erhaltenen Informationen für die Optimierung des eigenen Diagnoseprozesses ein. Als Technik der Transportschicht wird ein Diagnoseprotokoll vorgestellt, welches durch die Zuhilfenahme von Informationen niedrigerer Schichten in der Lage ist, fehlerhafte Links, Switches und Crossbar-Verbindungen zu lokalisieren. Als Machbarkeitsnachweis für das vorgeschlagene Interaktionsschema wird das Diagnoseprotokoll mit einer funktionalen und einer strukturellen Diagnosetechnik kombiniert und die

Kombinationen hinsichtlich ihrer Performanz und Diagnosequalität untersucht. Die Ergebnisse der Untersuchung zeigen, dass die kombinierten Techniken im Vergleich zu der funktionalen und der strukturellen Einzeltechnik eine deutlich reduzierte Fehlerlokalisierungslatenz besitzen. Dies geschieht jedoch auf Kosten der Diagnosequalität.

Der Fokus dieser Dissertation bezüglich Fehlertoleranz liegt auf dem Entwurf und der Umsetzung von schichtenübergreifenden Ansätzen, welche Software-Methoden einsetzen, um Fehlertoleranz für Routings der Vermittlungsschicht zu gewährleisten. Hierzu werden zwei Ansätze für unterschiedliche Routings vorgestellt.

Zunächst werden die Anforderungen an ein Network-on-Chip diskutiert, welche die Kommunikation von Informationen der unteren Schichten zu der Software mittels der vorhandenen Ressourcen und Schnittstellen erlauben. Hierfür werden die Konzepte zweier Mechanismen der Sicherungsschicht (data link layer) präsentiert, die die Umsetzung von Statusinformationen in kommunizierbare Einheiten sowie die Vermeidung der Blockierung von Kommunikationsressourcen gewährleisten. Im ersten Ansatz wird eine Software Rerouting-Methode vorgestellt, welche mittels der schichtenübergreifenden Interaktion Fehlertoleranz für ansonsten nicht-fehlertolerante, deterministische Routings der Vermittlungsschicht bietet. Als Spezialisierung der Rerouting-Methode kommt eine beispielhafte Umsetzung für Dimension-Order XY Routing zum Einsatz. Im zweiten Ansatz wird ein fehlertolerantes Routing für hierarchisch organisierte Network-on-Chip Topologien vorgeschlagen, bei welchem schichtenübergreifende Interaktion dazu eingesetzt wird, die autonome Verwaltung von hierarchischen Einheiten zu ermöglichen und das Routing zu rekonfigurieren. Die Ansätze werden hinsichtlich ihrer Performanz sowie ihrer Implementierungskosten evaluiert.

In einer abschließenden Betrachtung werden die schichtenübergreifenden Diagnosetechniken und Fehlertoleranz-Ansätze miteinander kombiniert und bezüglich ihres Einflusses auf die Performanz von Networks-on-Chip evaluiert. Hierfür dienen die Diagnoseinformationen als Grundlage für Rerouting und Routing-Rekonfiguration. Die Ergebnisse zeigen, dass bei übermäßigem schichtenübergreifenden Informationsaustausch Teile des Netzwerks überlastet werden können, was zu einer Reduktion der Performanz führt. Bei schichtenübergreifender Diagnose fällt jedoch der negative Einfluss auf die Performanz im Vergleich zu funktionaler und struktureller Diagnose deutlich geringer aus.

# Chapter 1
# Introduction and Motivation

Over the last decades, the integration density of chips has continuously increased as a result of the ongoing technology scaling. As a consequence, more and more complex on-chip systems have become feasible. Today's available manycore architectures feature up to one hundred cores [66]. It is predicted that until the year 2030 the feature size will come below 10 nm [67]. This enables future manycore systems with hundreds or even thousands of cores [94] [104] [108].

The strongly increasing number of cores on a chip, however, imposes new requirements on the communication architecture. From the design perspective, reusability and scalability of the communication infrastructure are of major concern [69] [94]. Additionally, a fast data communication between cores is required to gain a high system performance. For smaller systems classical bus-based communication works well, however, for future systems busses can no longer satisfy these requirements. Even for a small number of cores, the bus becomes the limiting factor of the system performance [69] [145]. Furthermore, in case of a bus failure, system components can no longer communicate with each other. This in turn may lead to a complete failure of the system.

To satisfy the requirements of future manycore systems, Networks-on-Chip (NoC) have been proposed as new design paradigm for on-chip communication [12] [31] [57]. NoCs enable parallel inter-core communication by applying the concept of macroscopic networks to on-chip systems. Similar to macroscopic networks, NoCs have a layered architecture as proposed by the OSI reference model [100]. Each of the layers represents a different abstraction level of a NoC.

While technology scaling is beneficial regarding the integration density, it also imposes new challenges for future on-chip systems. With the decreasing feature size, on-chip systems become more and more susceptible to hardware defects. These defects are introduced during chip manufacturing [13] or may be the result of wearout effects such as electromigration [85] during chip operation and are modeled by permanent faults [141].

Permanent faults compromise the functionality of a NoC. For instance, faults can lead to distortion of individual signals or may cause the functional misbehavior of NoC switches. This, in turn, results in an incorrect system behavior or, in worst case, in a complete system failure. To continue system operation correctly, it is crucial for NoCs to be able to locate permanent faults and to deal with them.

In the area of NoCs, a variety of fault diagnosis and fault tolerance approaches exists in literature. These approaches are associated with individual network layers and make use of the information available on the corresponding layer to deal with the fault. A permanent fault, however, may affect functionalities of different layers at the same time. Thus, it becomes necessary to deal with the fault's impact on more than just a single network layer. In general, it is possible to implement methods on various layers to cope with faults. However, without any interaction between the methods, actions performed on different layers may contradict each other. In addition, there is the necessity to repeat the same computations for the different methods, e.g. to localize the fault. For this purpose, it may also be required to replicate the corresponding hardware logic [18]. This results in a reduced performance and in an increased hardware overhead.

Because of these disadvantages, the demand for cross-layer interaction for fault diagnosis and fault tolerance has been made in literature for various areas of embedded systems [18] [37] [79] [105] [141]. Applying the concept of cross-layer interaction to the network abstraction layer model defined by OSI relaxes the strict separation of network layers by allowing the exchange of information across layer boundaries. This information exchange is expected to help to handle a fault with less effort than single layer approaches. Moreover, providing information of lower layers to higher ones allows methods to be shifted to the higher layers. Some methods may even be relocated from hardware to software resulting in a reduction of the hardware implementation costs.

Research on cross-layer design for fault diagnosis and fault tolerance in NoCs is in the early stages. Despite its expected advantages, only few approaches in literature exist so far that consider cross-layer design. Compared to single layer approaches, cross-layer design has additional challenges. In order to work cooperatively, the methods distributed over different network layers require coordination [105]. To this end, it is necessary to investigate possible cross-layer interactions and to specify the information flow across the layers. Moreover, the tradeoff between different parameters (e.g. hardware implementation costs or the latency to locate or deal with a fault) emerging from cross-layer design for fault diagnosis and fault tolerance have to be considered [141].

## 1.1 Contribution

This dissertation investigates cross-layer design for fault diagnosis and fault tolerance in NoCs with the focus on performance-related and quality-related parameters. The tradeoff between these parameters is studied.

This dissertation contributes to the state-of-the-art by the following:

1. A generic interaction scheme is proposed that combines protocol-based diagnosis on the transport layer with functional diagnosis and structural diagnosis techniques. This interaction scheme can be used to increase the overall performance of diagnosis by efficiently determining the faulty switch on the transport layer before applying fine grained diagnosis of the lower layer techniques [121].
2. For diagnosis on the transport layer a diagnosis protocol is proposed that is able to narrow down a fault on a communication path to a single link, switch, or crossbar connection [118].
3. The performance and diagnosis quality of cross-layer diagnosis is investigated for combinations of protocol-based diagnosis with functional and structural diagnosis and the results are compared to the corresponding standalone techniques [121].
4. The need for lower layer mechanisms to manage communication resources is discussed. Concepts for two hardware mechanisms of the data link layer are presented to avoid communication resources from being blocked and to provide information about the availability of communication resources to higher network layers.
5. A software-based rerouting approach is proposed that, with the help of information of lower network layers, is able to offer fault tolerance for otherwise non fault tolerant, deterministic network layer routings [118]. An exemplary implementation of the method for dimension-order XY routing is presented.
6. A reconfigurable routing approach for hierarchically organized NoCs [119] with a cross-layer reconfiguration process [117] is proposed that allows the routing within individual hierarchical units to be adapted while the rest of the NoC remains operative.

## 1.2 Dissertation Outline

This thesis is organized as follows: Chapter 2 introduces the basics of NoCs relevant for this work. An overview of state-of-the-art and related work covering diagnosis and fault tolerance methods is given in Chapter 3. In Chapter 4, cross-layer localization of permanent faults is discussed. First, the interaction scheme to combine protocol-based diagnosis of the transport layer with functional and structural diagnosis of lower layers is presented. Then, the diagnosis protocol to locate faulty communication resources in a NoC is introduced. The protocol is combined with a functional and a structural diagnosis technique according to the proposed interaction scheme and all combinations as well as the corresponding standalone diagnosis techniques are evaluated with respect to the diagnosis quality and localization latency. Chapter 5 covers cross-layer fault tolerance. First, the two data link layer mechanisms for flow control management and for providing availability information of communication resources to the software are discussed. Following this, the software-based rerouting and the reconfigurable routing for hierarchically organized NoCs are presented and are evaluated. The combinations of diagnosis techniques and fault tolerance methods are evaluated in Chapter 6. Chapter 7 summarizes this dissertation and proposes possible future work.

# Chapter 2
# Preliminaries

## 2.1 Networks-on-Chip

Networks-on-Chip (NoC) have been proposed by various research groups [12] [31] [57] around the year 2000 as new communication architecture for future manycore systems. NoCs are scalable on-chip communication networks inspired by macroscopic networks. They allow the parallel exchange of information between cores using a predominantly packet-based communication. Because of their inherent redundancy provided by multiple alternative paths between two cores, NoCs have the potential to tolerate faults on a communication path. Similar to macroscopic networks, the architecture of NoCs follows the concept of network layers [38] [86] [91] as defined by the *Open Systems Interconnection* model (OSI) [100]. The OSI network stack comprises seven network abstraction layers as shown in Figure 2.1. Because of their general

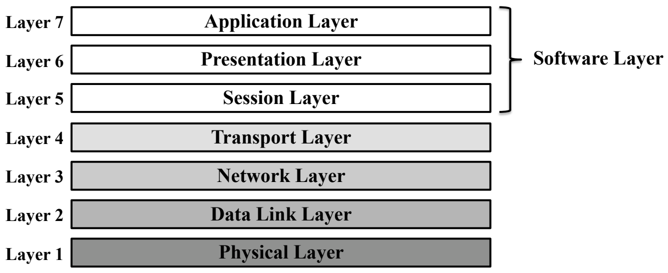


Fig. 2.1: OSI network stack.

character, the layers five to seven, i.e. session layer, presentation layer, and application layer, are not NoC specific. These three layers are subsumed as *software layer* in this thesis. The implementation of the functionality and the communication mechanisms of the network layers two to four are NoC specific and are subject to NoC specific requirements such as minimizing implementation costs. The transmission of information on the physical layer again is general for all kinds of on-chip systems. However, there also exists NoC-related research focusing on the physical layer, e.g. the wireless interface in the area of wireless NoCs [35] [36]. In the following, a layer-wise overview is given about significant NoC components and the functionality of each network layer.

### 2.1.1 Software Layer

The software layer is an application-oriented layer. It provides services and protocols to the application for the data exchange via the underlying communication architecture. In the network stack with strictly separated layers as defined by OSI, the communication architecture type, e.g. point-to-point bus or NoC architecture, as well as the functionality used for the communication on lower layers is hidden from the software layer. Software layer related research for NoCs covers optimized application programming interfaces (API) [75] as well as optimal task scheduling [143] and mapping [53] [113] of applications to cores.

Cores are computational components of the software layer executing the applications and implementing the aforementioned services and protocols. Besides e.g. arithmetical hardware units, cores belong to the class of *processing elements* (PE).

In this thesis, the term PE refers to a core in combination with local memory. PEs are used as sources and sinks of communication. Neither APIs nor application mapping is part of this thesis.

### 2.1.2 Transport Layer

The transport layer constitutes the interface between the PE and the network switch (cf. Subsection 2.1.3). For this purpose, NoCs are equipped with network interfaces (NI) that provide communication protocol services and network services [34] allowing data to be passed from software layer to the network and vice versa. As mentioned previously, the data communication in NoCs is typically packet-based. Packetization and depacketization of data are important services offered by the NI. As the size of packets normally exceeds the width of communication resources in the NoC, i.e. the width of buffers and communication channels, packets are further subdivided into *flow control* *units* (flits). Each packet is composed of a *head* flit containing the routing information and a *tail* flit defining the end of a packet. Between *head* flit and *tail* flit an arbitrary number of *body* flits exist that contain the payload data.

Further important tasks of the transport layer are end-to-end flow control [20] and error recovery [95]. Flow control regulates the communication flow between two PEs and is responsible to allocate enough memory to store a packet. End-to-end error recovery ensures reliable communication between two communication endpoints. If a packet is corrupted or lost the corresponding packet is retransmitted by its source. In order to detect packet corruption, all packets are commonly equipped with *error detection* or *error correction* codes and are checked at the receiver.

### *2.1.3 Network Layer*

The network layer's task is the transport of packets from their source NI to their destination NI. The required functionality for transportation comprises:

- the routing,
- the arbitration, and
- the switching of packets [34].

This functionality is encapsulated in network switches (SW). A switch has input ports *i* and output ports *o* to receive and forward packets. In the following, the term *port* refers to an input/output port pair. The local port of a switch is connected to the NI. To be able to store flits or entire packets temporarily, switches may implement input or output buffers at each port. Figure 2.2 shows an exemplary five-port switch with input buffers.

Routing defines the path of data through the network from its source to its destination. At every switch, the flits of a packet received on an input port have to be forwarded to an output port. The set of possible output ports is determined by the routing. As routing input parameter, the destination address stored in the head flit of a packet is used. To provide routing functionality, every switch implements a router component. The complexity of this component depends on the routing type used (cf. Subsection 2.2).

As several packets at different input ports may request for the same output port *o* at a time, a switch implements arbitration functionality for each output port. From the set of all requests, arbitration selects one of the requests and gives the grant for the output port to the corresponding input according to the

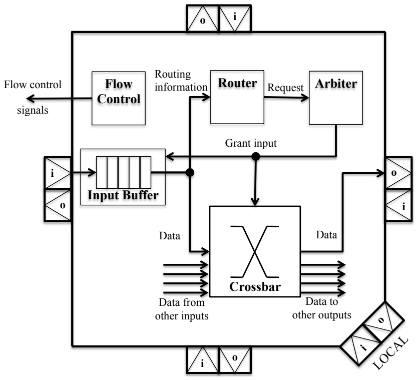


Fig. 2.2: Input-buffered five-port switch.

arbitration policy, e.g. *round robin* arbitration. The flits of a granted input traverse the switch-internal *crossbar* towards the output. A crossbar connection connects an input port $i$ to an output port $o$ of a switch. In the following, a crossbar connection is denoted as $i \to o$.

Switching defines how data is transferred through a NoC. Circuit switching and packet switching methods exist, packet switching being the more common for NoCs [34]. The circuit switching method first reserves the whole path between source and destination before data is sent. Packet switching, on the other hand, does not reserve a path in advance and each packet may use another path through the network. Packet switching methods differ with regard to the granularity of how data is forwarded and how buffer space is allocated. Data can either be forwarded in packet or flit granularity. Packet granularity requires that the entire packet can be stored at the next switch before the packet is forwarded. Buffer space is allocated based on the packet size. In case of flit granularity, buffer space is allocated based on the flit size.

A packet switching method with flit granularity is *wormhole switching* [28]. In wormhole switching, the head flit reserves an output port at each switch it passes, and thus, it reserves the path of a packet. Body and tail flits just follow the head flit and the tail flit cancels the reservations. A flit is forwarded as soon as the input buffer of the next receiving switch has a free buffer slot to store it. This allows the capacity of input buffers to be reduced to only one buffer slot. Compared to other packet switching methods, i.e. *store and forward* (SAF) or *virtual cut through* (VCT), wormhole switching has the benefit of a smaller packet latency at each switch (SAF) and a smaller overall buffer implementation overhead (VCT) [34] [97]. This makes wormhole switching attractive to NoCs [16] [87] and, as a consequence, it is used as switching method in this thesis.

For the communication of data packets, bidirectional communication links are used that connect two switches to each other through their ports. A link consists of two unidirectional channels leading in opposite directions. In this thesis, in addition to links, the crossbar connections from the input ports of a switch to its output ports are considered to be part of the network layer as well.

The topology of a NoC results from the interconnection structure of the network switches. Because of its regular and planar structure, the *mesh* is the most common topology for NoCs found in literature. An exemplary 3x3 NoC with mesh topology is shown in Figure 2.3.

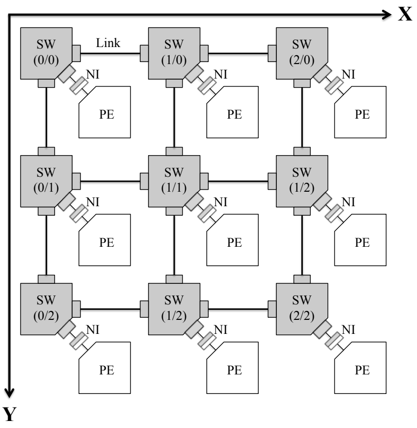


Fig. 2.3: 3x3 NoC with mesh topology.

NoCs can be classified as *direct* or *indirect* networks. In contrast to indirect networks that contain switches without connection to a PE, in direct networks each switch is connected to a PE (cf. Figure 2.3) [28]. A NoC is *homogeneous* if all PEs are of the same kind. Otherwise it is *heterogeneous* [73].

In this thesis, NoCs with a direct and homogeneous topology are considered and each PE corresponds to a core. The combination of switch, NI, and PE is called a *network node*. To distinguish between the different nodes, each one has a unique identifier (ID) used to address it.

### *2.1.4 Data Link Layer*

The data link layer is responsible for the reliable communication of packets via links. While on the network layer links are abstracted as a logical interconnection component between switches, on the data link layer the individual wires of a link are considered. Reliable communication comprises the protection of data against faults as well as the allocation of switch resources. Both are tasks of flow control (cf. Figure 2.2).

Reliable communication can be ensured on the data link layer by means of switch-to-switch error recovery (s2s). In contrast to end-to-end error recovery (e2e) of the transport layer, s2s requires at least one additional packet checking hardware unit in each switch. The receipt of packets is positively or negatively acknowledged by the receiving switch. If a packet is faulty, it is retransmitted on a switch-to-switch basis. This implies, however, that each switch has to store packets until they are correctly forwarded to the next switch. Although, s2s typically results in a smaller average packet latency than e2e, it has a higher power consumption and a higher hardware implementation cost for buffers [95]. For this reason, in this thesis e2e recovery is used.

Packets traversing the NoC occupy resources of a switch such as buffer slots. A packet or a single flit may only be forwarded via a link if the receiving switch has enough free resources to accept it. If not enough resources are available at the receiving switch, the flow control mechanism stalls the communication until enough resources are free again. While some flow control mechanisms such as *ACK/NACK* inherently support error recovery (s2s), for *credit-based* flow control error recovery has to be additionally provided on the transport layer (e2e) [34]. In this thesis, credit-based flow control is used in combination with *Selective Repeat ARQ* [137] error recovery on the transport layer.

Switches with credit-based flow control keep track of the free buffer slots of the corresponding neighbor switches by means of counters. For this purpose, at every output port of a switch, a counter is implemented whose initial value equals to the number of available buffer slots at the neighbor's input port. The counter is decreased each time data is forwarded to the neighbor switch and is stored there in the buffer. If the neighbor switch further forwards the data, i.e. the corresponding buffer slot becomes free again, this is signaled back to the sending switch which increases its respective counter.

In this thesis, the control signals, e.g. grant signals, used to implement higher layer functionality such as switching are considered to be part of the data link layer as well.

### 2.1.5 Physical Layer

The lowest of the layers defined by OSI is the physical layer. It deals with the physical implementation and electrical characteristics of communication channels as well as the physical transfer of data via communication channels. To match with the width of physical channels, flits are further divided into *physical digits* (phits). For the transfer of phits, issues such as the signal integrity, low-power signaling, and the synchronization of signals are of concern [34]. These issues, however, are not specific to NoCs but are of general importance for on-chip systems.

The physical layer, as defined by the OSI model, only considers the above issues for interconnects. In this thesis, however, the wires and digital logic gates of switch logic are attributed to the physical layer as well.

### 2.1.6 Definition Cross-Layer

In the network stack defined by the OSI model, the layers are strictly separated from each other and each layer implements communication services, e.g. transfer of data between two network switches, together with the respective functions, e.g. routing [100]. A layer only accesses the services of next lower layer. For this purpose, each layer offers service interfaces to the next higher layer. The services are used for the exchange of data between the two endpoints of a communication. The access to services of a non adjacent layer or the access to layer-specific information, e.g. which routing algorithm is used on the network layer, is not provided by the OSI model.

In the cross-layer design the strict layer boundaries are dissolved, allowing layers to exchange information [128]. The information exchange can be either *top-down* or *bottom-up* . In a top-down flow, information of a higher layer is provided to one or more lower layers whereas in the bottom-up flow it is the other way around.

Cross-layer design constitutes the base for multi-layer diagnosis and fault tolerance approaches where different mechanisms or parts of one mechanism are implemented on different network layers. Thanks to the exchange of information between layers the time required for fault localization [105] or reconfiguration [42] is reduced. Furthermore, by making use of information of different layers the implementation overhead can be decreased [18] [79].

In this thesis, the cross-layer information exchange is used to combine diagnosis techniques of different network layers and to enable fault tolerance to be performed in software.

### *2.1.7 Hierarchical Networks-on-Chip*

The increasing size of NoCs leads to scalability issues for mechanisms implemented in the NoC such as table-based routing or fault diagnosis. A possibility to cope with these scalability issues is to design NoCs hierarchically and to apply global mechanisms to smaller hierarchical units.

A hierarchical NoC can be designed by either constructing its topology by means of physically separated subnetworks [17] [33] [62] [61] [107] or by segmenting a given topology into logical units [106] [140]. To construct a hierarchical topology using subnetworks, it is required that the they are interconnected by at least one superordinated network. Logical grouping results in a logical unit. It requires the segmentation of an existing topology. In contrast to subnetworks, in which a network node belongs to one physical subnetwork, nodes can be part of multiple logical units. Typically, network nodes are grouped into a logical unit if they are part of the same task and share a spatial relation.

For the reconfigurable hierarchical routing presented in Section 5.3 the NoC topology is segmented into logical hierarchical units.

## 2.2 Routing

### *2.2.1 Classification*

Routing methods can either be implemented in a *distributed* or in a *centralized* manner. In the former case, the path of a packet evolves incrementally per switch, and thus, every switch requires a router component implementing the routing method either as combinational logic or by storing the routing decisions in routing tables. In centralized routing methods, also referred to as source-based routing methods, the packet's path through the network is predefined at sender side and all routing decisions are added to the packet header. In this case, the functionality of the router component at each switch is reduced to looking up the routing decision stored in the packets.

A routing method can be further categorized as *deterministic* or *adaptive*. For a given source and destination pair, deterministic routing methods always provide exactly the same path regardless of the network status. A typical example for a deterministic routing method is *dimension-order* routing (cf. Subsection 2.2.4). Routing methods that do not consider the network state are also called *oblivious* routing methods. Apart from the special case of deterministic routing methods, other oblivious routing methods may support multiple paths between a source and a destination, however, the path is randomly picked.

Adaptive routings methods take the network status into account allowing them to react on network changes [28]. This, for instance, enables the rerouting of packets to alternative paths in case of congestion of the original path. Adaptivity is the essential requirement for routing methods in order to tolerate the failure of network components. Adaptive routing methods can be further categorized as partially or fully adaptive. While partially adaptive routing methods can only tolerate the failure of components to some extent, fully adaptive methods offer connectivity between all network nodes as long as the topology is connected.

Routings can be further classified as *reconfigurable* routings. In contrast to adaptive routings, reconfigurable routings are not necessarily able to automatically tolerate a change of the network's topology. However, reconfigurable routings allow the recalculation of routing for the altered topology. Once the recalculation process is finished, the routing is updated in the network. A reconfigurable routing for a hierarchically organized NoC topology is presented in Section 5.3.

An important requirement to all routing methods is that they do not cause a deadlock in a NoC.

### *2.2.2 Deadlocks*

A *deadlock* describes the situation in which packets are stuck in the NoC and cannot be forwarded further. The reason of a deadlock can either be message-dependent or routing-dependent.

Message-dependent deadlocks emerge on the transport layer or higher layers. They are caused by dependencies of different message types, such as requests and responses, sharing the same communication resources [96]. If the

NoC is congested by request messages, no responses of already processed requests can be injected to the network by PEs. A PE has a message buffer of limited size. If this buffer is completely filled with requests and at the same time no response can be injected, no further requests can be accepted by the PE. When the response cannot be injected due to the network congestion, the next request cannot be removed from the buffer. Thus, no buffer slot becomes free and no further requests can be accepted by the PE. A way to cope with this issue is to separate communication resources for each message type [96].

Routing-dependent deadlocks are a problem of the network layer. The cause of deadlocks are cyclic dependencies between the communication resources of a NoC. For instance, if a packet $A$, in order to move forward, requires the communication resource held by a packet $B$ and, at the same time, $B$ requires the resource held by $A$, both packets are stuck. Such cyclic dependencies are commonly eliminated by routing restrictions, e.g. forbidden turns a packet may take in a NoC, or by a routing policy. Routing methods with no restrictions or no appropriate policy are not deadlock-free. If a routing method is not inherently deadlock-free, virtual channels can be used to ensure deadlock-freedom (cf. Subsection 2.2.3).

Deadlock-freedom is of great importance to fault tolerance where a routing method has to be reconfigured to bypass faults. Deadlocks may also occur during the reconfiguration process if the NoC is e.g. only partially reconfigured and packets are communicated using the old and the new routing [92]. Thus, it has to be ensured, that deadlock-freedom is guaranteed at any time of the routing reconfiguration process.

### 2.2.3 Virtual Channels

Communication channels (cf. Subsection 2.1.3) may be further logically subdivided into *virtual channels* (VC). For this purpose, two or more VCs are multiplexed on a channel. For each VC of a channel, a switch has to implement a VC buffer. Flits using a VC are stored in the corresponding VC buffer.

Originally, VCs have been proposed to avoid routing-dependent deadlocks in multiprocessor interconnect networks [30]. By defining a total order on the virtual channels and allowing them to be allocated in either ascending or descending order only, cyclic channel dependencies are resolved. The concept of VCs has been adopted to increase network performance by avoiding head of line blocking [29].

In this thesis, VCs are used to ensure deadlock-freedom for the reconfigurable hierarchical routing (cf. Section 5.3).

### *2.2.4 Dimension-order Routing*

A typical example for oblivious routing is dimension-order routing (DOR) where coordinates are used as switch addresses. A well known DOR routing used for the two dimensional mesh topology is XY routing, which is restricted to allow x-to-y turns only. A packet is first sent into x-direction until the x-address of the current switch equals the one of the destination switch. Then the packet is forwarded into y-direction until the destination is reached. This routing rule prevents any cyclic dependencies, and thus, XY routing is inherently deadlock-free without the use of VCs.

For the proposed software-based packet rerouting approach in Subsection 5.2 an implementation for dimension-order XY routing is presented.

### *2.2.5 Up/Down Routing*

*Up/Down* routing method, as originally proposed in [123], allows the calculation of an inherently deadlock-free routing for arbitrary network topologies. It is assumed that each network node has a unique identification number (ID). For the routing calculation for a given topology, first, the spanning tree of this topology is created by a breadth-first search starting from an arbitrarily chosen root node. Based on the spanning tree, *Up* or *Down* direction is assigned to each communication channel $c$.

For two connected network nodes $n_i$ and $n_j$, *Up* direction is assigned to channel $c_{i,j}$ if the distance of node $n_j$ to the root node of the spanning tree is less than the one of $n_i$. If two network nodes have the same distance, the channel leading to the node with the smaller ID becomes the *Up* channel. In all other cases, *Down* direction is assigned.

To ensure deadlock-freedom of *Up/Down* routing, a valid path must consist of zero or more channels in *Up* direction followed by zero or more channels in *Down* direction [123]. The change in a path from an *Up* to a *Down* channel is called an *Up* to *Down* turn. *Down* to *Up* turns are not allowed.

In this thesis, *Up/Down* routing is used as the basis of the reconfigurable hierarchical routing proposed in Subsection 5.3. Native *Up/Down* routing does not require any VCs for deadlock-freedom. VCs are required, however, for the reconfigurable hierarchical routing in order to allow routing calculation to be performed independently for each hierarchical unit in the NoC.

### *2.2.6 Determination of Shortest Path*

A common issue for the calculation of routings is the determination of the shortest path between two communication endpoints. Three well-known algorithms to determine shortest paths in graphs are the Bellmann-Ford [11], Floyd-Warshall [46], and Dijkstra [40] algorithms. The latter one is used as basis in this thesis for the calculation of the hierarchical routing (cf. Subsection 5.3).

The Dijkstra algorithm as described in [40] finds the shortest path between a given start and destination vertex in a graph with non-negative edge weights [89]. Beginning at the start vertex, at each visited vertex the algorithm discovers all the reachable adjacent vertices and calculates the costs to reach them. The cost of a discovered vertex is composed of the cost of the currently visited vertex plus the weight of the edge. Out of the set of all discovered vertices always the vertex with the least cost is visited next. The algorithm either terminates as soon as the destination vertex is reached or when all vertices have been visited. The second variant is used in this thesis.

## 2.3 Faults and Fault Models

### *2.3.1 Fault Types*

Defects as well as external influences due to radiation physically affect transistors and wires on a chip causing them to operate outside their specification or to fail. The failure or the malfunctioning of transistors and wires is modeled by a fault [108]. The observable manifestation of a fault such as a signal change is called an error [77] [136].

In general, faults are classified as the following three fault types:

- transient,
- intermittent, and
- permanent.

Transient faults only appear for a short period of time at random locations before they disappear again. They are predominantly caused by neutrons from cosmic rays and by alpha particles. When a particle hits the chip, it induces a charge that can lead to soft errors such as a bit flip of a memory cell or an incorrect value of a logic gate [108]. The typical occurrence frequency of soft errors, i.e. *soft error rate* (SER), for a chip is $10^{-9}\ 1/s$ [34]. However, investigations have shown that the SER, increases with decreasing feature sizes of chips [10] [13] [93] [135].

If the same fault continuously appears and disappears in regular or irregular periods at the same location, this fault is called an intermittent fault. The manifestation of errors induced by intermittent faults is similar to the one of errors caused by transient faults [21]. In contrast to transient faults, however, intermittent faults are not caused by external disturbances but by aging effects and structural defects. Another cause for intermittent faults is crosstalk due to coupling effects between adjacent wires [47] [135]. The occurrence of an error and thus its observability depends on the operation condition of a chip. An error may become observable only for a particular supply voltage or temperature or if the system is operated outside the specification [21] [138]. Eventually, an intermittent fault may result in a permanent fault [108].

Permanent faults are caused by stationary structural chip defects, whose impact, in contrast to the other two fault types, does not disappear again. Structural defects may occur during chip manufacturing or during system operation. Manufacturing inaccuracies, e.g. during lithography, may cause wires to be broken or result in shorts between neighbored wires by additional material [135]. Further challenges for chip manufacturing are process variations and parameter variations [14]. Process variations can change the geometry of wires and transistors and thus lead to changes in delays or the electrical characteristic of these elements [13] [15] [135]. Variations can lead to a loss of system performance or to a permanent malfunction of elements [101].

During system operation permanent faults are caused by aging effects such as electromigration: the momentum of moving electrons in a wire induces a force causing metal atoms to be released from their lattice and to be transported in the direction of the electron flux. The removing of metal atoms results in voids in a wire increasing the wire delay and resulting in a broken wire eventually. Elsewhere, these atoms accumulate and the additional material may result in shorts between neighboring wires [85] [124] .

Each of the three fault types can lead to errors. In turn, these errors result in different failures on the different network layers. An example of possible failures for each network layer is given in Table 2.1. In the worst case, each

Table 2.1: Failure Examples

| Layer | Failure |
|---|---|
| Software Layer | wrong computation |
| Transport Layer | flow control failure |
| Network Layer | packet misrouting |
| Data Link Layer | misallocation of resources |
| Physical Layer | logic circuit misbehavior |

of these failures can result in a malfunction of the system. For this reason, in order to continue system operation in the presence of faults, systems have to be designed to tolerate faults on different network layers. For the design, the impact of a fault on the NoC is captured by a so called fault model (cf. Subsection 2.3.2).

The handling of transient faults as well as permanent faults is important for NoCs. Because transient faults disappear after a short period of time, rather simple methods such as retransmitting faulty data can be used to tolerate them. In case of permanent faults, more complex methods are necessary in order to locate their position in the NoC and to ensure NoC operation to be continued. Some methods presented in this thesis are capable of handling transient faults (cf. Subsection 4.2.1 and Subsection 5.1.2). Because of their complexity, however, the focus of the proposed cross-layer methods is on localizing and tolerating permanent faults. Intermittent faults are not considered due to their similarities to transient and permanent faults.

### *2.3.2 Fault Models*

In order to design fault tolerant NoCs, *fault models* are used that represent the impact of complex effects caused e.g. by neutron strikes or electromigration on a NoC by means of an abstracted model [34] (p. 81). These models constitute the base for the design of fault detection, fault diagnosis, and fault tolerance methods. Faults not considered in the model are not guaranteed to be covered by a method. However, it is possible that unconsidered faults may be covered by a model to some extend when they have similar characteristics as the considered faults.

Due to the random occurrence of transient faults, they are commonly modeled by means of a *soft error rate* (SER) or a *bit error rate* (BER) [5]. The SER models the probability of occurrence of soft errors in the NoC in general. Depending on the targeted granularity, fault models for transient faults model the impact of soft errors on communication resources, packets, or switch functions such as routing. In contrast to SER, BER is related to data only. It states the rate at which data bits are corrupted during communication [135], and is modeled by a bit flip.

Fault models for intermittent faults cover errors caused by wearout effects as well as caused by noise such as crosstalk. To model wearout and manufacturing defect induced errors on communication interconnections, delay, open, and short fault models are used [55]. Crosstalk is of great concern for NoCs [108] and different fault models exist that represent crosstalk errors such as delays, speed up, or glitches on links [56] [127].

Available fault models for permanent faults model the failure of components at different levels of abstraction and granularity. The focus of fault models used for NoCs is on the data link, the network, and the transport layer. A model widely used for permanent defects in single data or control signals of a link or crossbar connection on the data link layer is the stuck-at model. In this model, the value of a signal is permanently forced to either logic 0 or logic 1 [136]. In contrast to the fine granularity of data link layer fault models, fault models related to the network layer assume the failure of network components such as links, crossbar connections, or entire switches [5] [41]. Furthermore, functional fault models exist that describe the misbehavior of a switch function such as misrouting [27]. Transport layer related models consider the failure of NIs [80] and the corruption or loss of packets [5].

In this thesis, fault models for transient and permanent faults are used. Similar to what is described above, transient faults are modeled by a probability for data corruption and loss. The models for permanent faults assumed for the diagnosis and fault tolerance approaches comprise the failure of communication resources such as crossbar connections, links, and complete switches. In addition to erroneous data, the corruption of signals related to flow control are considered as well. Transient faults are modeled by means of a bit-flip, while for permanent faults the stuck-at model is used.

The underlying fault model of each approach is presented at the beginning of the corresponding section.

## 2.4 Fault Tolerant Networks-on-Chip

A system that is able to continue correct system operation in the presence of faults is called a *fault tolerant system*. In order to continue operation a fault tolerant system in general requires the ability to:

- detect errors,
- to locate faults, and
- to cope with faults [28] (p. 516).

This applies to NoCs as well. Mechanisms to detect errors, to locate the underlying faults, and to tolerate the faults can be implemented on the different network layers. For instance, the detection of errors can be implemented on the network layer as dedicated hardware checking units [74], on the transport layer as software checks [5], or can be a combination of both [95]. While error detection and fault tolerance is of concern for all types of errors or faults, respectively, the localization is only necessary for permanent faults.

A common method to detect errors caused by transient and permanent faults in NoCs is to equip packets with *error detecting codes* (EDC) or *error correcting codes* (ECC) [34] [81] [112] [146]. Both, EDC and ECC, add redundancy to packets by adding a checksum or by transforming the payload into code words. If the checksum or code word of a received packet is wrong or unknown, this indicates that the packet was altered by a fault. Another method to detect errors is testing NoC components by means of *built-in self-test* (BIST) hardware units [22] [43] [56] or by applying dedicated test patterns to the inputs of a component and analyzing the resulting test responses at the outputs [26]. In order to apply test patterns to NoC switches, a *test access mechanism* (TAM) is required [7] [63] such as additional scan chains [135] [139]. For NoCs *structural* as well as *functional* test strategies are available to detect errors in NI [130], switch [7] [70], and link components [59].

When an error is detected, the type of the underlying fault has to be determined and, in case of a permanent fault, its position in the network has to be located to identify the affected NoC component or the affected part of a component. Identifying the type of fault and localizing its position are the tasks of *fault diagnosis* [108] [136]. For the localization of faults, fault diagnosis analyzes the results of the preceding tests [141]. Similar to error detection, fault diagnosis techniques can be employed on different network layers using different network information as basis of their diagnosis process, as shown in Figure 2.4.

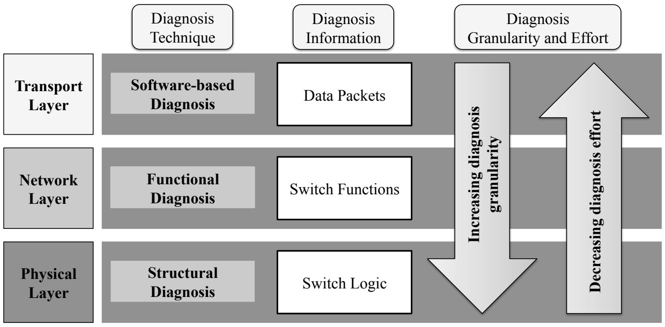


Fig. 2.4: Diagnosis techniques.

The diagnosis techniques considered in this thesis are:

- *Protocol-based Diagnosis* on the transport layer,
- *Functional Diagnosis* on the network layer, and
- *Structural Diagnosis* on the physical layer.

Each of the three techniques differs with regard to the used diagnosis information, the diagnosis granularity, and the diagnosis effort.

After the localization of a fault, appropriate measures have to be initiated to cope with the fault and to allow the NoC communication to be continued. In literature a wide range of fault tolerant approaches of different network layers exist. In general, these approaches can be divided into methods for transient faults and methods for permanent faults. Due to the similarity of intermittent faults and permanent faults regarding their manifestation, fault tolerance methods to cope with permanent faults can be applied for intermittent faults as well [108]. As with diagnosis, the methods to tolerate faults can be classified according to the network layers. The layer of a fault tolerance method is defined by the layer on which the method's measure is effective.

This thesis covers cross-layer fault localization and fault tolerance methods for permanent faults. The detection of errors is out of scope. However, it is assumed throughout the whole work, that packets are equipped with an EDC to detect the corruption of data on an end-to-end basis.

# Chapter 3
# Background and Related Work

In this chapter an overview of state-of-the-art fault localization methods (cf. Section 3.1) and fault tolerance methods (cf. Section 3.2) related to this thesis is given. The focus is on permanent faults (cf. Subsection 2.3.1). Single layer as well as cross-layer techniques / methods are included.

## 3.1 Fault Localization

For the localization of permanent faults in NoCs, diagnosis techniques on different network layers can be used. These techniques differ with regard to their diagnosis granularity and the required effort to locate the fault's position (cf. Figure 2.4). The granularity of a technique defines the accuracy with which a fault can be diagnosed. In general, the higher the network layer, the higher the degree of abstraction of the underlying hardware and, as a consequence, the less detailed information about the hardware is available, which can be used by diagnosis techniques. Thus, the diagnosis granularity of techniques of higher layers is less than the granularity of lower layer techniques. The increased granularity, however, comes at the expense of an increased diagnosis effort such as an implementation overhead due to additional hardware.

Details about the granularity and effort of structural diagnosis on the physical layer, functional diagnosis on the data link / network layer, and protocol-based diagnosis on the transport layer are presented in the following subsections (cf. Subsection 3.1.1 to Subsection 3.1.3).

### *3.1.1 Structural Diagnosis*

The diagnosis technique with the most fine-grained granularity is the structural diagnosis on the physical layer. It is capable of determining faulty gate and wire components in links and switches of a NoC. To locate a fault, first, the entire NoC has to be tested to identify the faulty component. Subsequently, the component has to be diagnosed.

In order to gain information about the fault state of components, additional hardware test structures have to be provided in the NoC design. Scan design is the methodology most widely used for structural testing. It allows test stimuli to be externally applied to the sequential logic of e.g. switches via so called scan chains [136] [139]. For this purpose, the switch has to be set to a special test mode where the scan chains shift in the applied test stimuli [136]. As test stimuli, test patterns are used to test NoC components e.g. for stuck-at [43], crosstalk [56], or bridging faults [59]. Test patterns can be generated by means of *Automatic Test Pattern Generation* (ATPG) algorithms [136]. With an appropriate test pattern set, structural testing achieves a high fault coverage from about 95% up to 100% [19] [22] [26] [43].

To transport test patterns to switches of a NoC and to transport the corresponding test responses to their sink, e.g. a diagnosis unit, in addition to scan chain further TAMs are used [7] [63] [135]. Depending on the test strategy, their implementation can result in a significant area overhead [7]. To reduce the overhead for NoCs, the reuse of the available communication architecture as TAM is proposed as a cost effective solution to test switches [6] [23] [56] [58] [59] [116]. To enable the reuse of the communication architecture for testing, switches have either to be set to a dedicated test mode to accept test patterns [116] or the test patterns have to correspond to the packet format used in the NoC [26] [56] [59] [83]. In order to keep the overall test time minimal, the available TAM approaches for NoCs allow several [63] [116] or all [7] switches to be accessed in parallel.

After completion of structural test, the test responses give information about the presence of a fault. At the physical layer, a common strategy to deal with faults is to shut down the affected component [141]. A fault in a NoC, however, does not necessarily affect the functionality of the entire link or switch, and thus, the complete shutdown of the respective component is inappropriate as it leads to unnecessary loss of NoC performance. Structural diagnosis locates the faulty gates in switches in order to determine the affected functionality of a component [25]. For the localization of faulty gates structural diagnosis analyzes the test responses. Depending on the fault model, structural diagnosis is able to localize between 60% and 85% of the faulty gates and wires of a logic circuit [22].

Structural diagnosis for NoCs can either be performed by *Automated Test Equipment* (ATE) or by *Built-In Self-Test* (BIST) [141]. ATE is an external test equipment used to test and diagnose chips after production for structural defects. For this purpose, ATE implements all the required test methods and the diagnosis functionality. Faults emerging during NoC operation, however, cannot be diagnosed by ATE. In contrast to ATE, BIST allows test and diagnosis of links [19] [56] [59], buffers [54], and whole switches [43] to be performed in-field. However, in-field fault localization requires the test pattern generation and test response analysis functionality to be implemented in the NoC, and thus, further increase the area overhead [19] [22] [43] [56]. This overhead can be reduced, for instance, by sharing one test pattern generator for the entire NoC [56] or by implementing test pattern generation and response analysis as functionality of PEs [26].

The time required by structural diagnosis to localize a fault is composed of the time for testing and the time for the actual diagnosis step. Besides the NoC size, the required time depends on aspects like the underlying considered fault model, the number of test patterns, if diagnosis is performed by BIST or by ATE, and whether switches can be tested and diagnosed in parallel or not. While the time for fault testing ranges from a few thousands of cycles [26] to tens of thousands of cycles [7], the required time for test plus diagnosis can increase to hundreds of thousands of cycles [43] [56].

In this thesis, the approach proposed by Dalirsani et al. [25] is used as representative of structural diagnosis on the physical layer.

### 3.1.2 Functional Diagnosis

In contrast to the fault models used to represent the effects of structural defects on the physical layer, the fault models of data link and network layer exhibit a higher degree of abstraction. As a consequence, the localization accuracy of diagnosis approaches of these layers decreases compared to structural diagnosis. For instance, available models for NoCs comprise the failure of links and switches, e.g. [41] . However, the failure of an entire switch due to a structural defect is a rather pessimistic assumption [108]. Functional fault models on the data link layer (links) and the network layer (switches) consider the failure of only a part of a link's or switch's functionality. These models provide the basis for functional diagnosis.

For testing the functionality of NoC components, functional diagnosis makes use of functional test patterns [27] [70]. In case of the presence of a structural defect these patterns cause an observable functional misbehavior of a component. For the generation of suitable patterns, the impact of structural defects in the data or control path on the functionality of a link or switch has to be characterized first. The impact is captured by *functional failure modes*

[141]. Common failure modes used in the field of NoCs for links and switches are corruption, loss, or delay of flits or packets [27] [52] [70] [71] [109]. For switches additional failure modes exist that comprise duplication of flits or packets, misrouting, deadlock and livelocks, and starvation [1] [27] [52] [70] [71]. Functional test patterns can be generated at design time [27] or online using test pattern generators [70]. Compared to structural diagnosis, functional diagnosis typically has a reduced pattern set [23] [50]. Instead of test patterns, some NoC approaches make use of normal data packets for testing [1] [52] [71].

In contrast to structural diagnosis, functional diagnosis does not use additional TAM or scan chains. Instead, test patterns are transported to components using the NoC's communication infrastructure and they are applied to the components via their standard communication interfaces. Hence, functional diagnosis has a smaller area overhead than structural diagnosis. Some failure modes, however, require additional hardware in order to become observable, e.g. hardware timers to observe starvation of packets [52], or to collect information necessary for diagnosis [1].

The smaller area overhead and reduced pattern set, however, come at the expense of a reduced structural fault coverage. In general, only faults are testable that are covered by at least one of the failure modes [135]. However, by means of satisfiability solver (SAT) based classification algorithm it is possible to map structural faults to failure modes and to generate appropriate functional test patterns to test switches [27]. As a result of this, a coverage of about 80% of structural faults is achieved.

While in prior approaches the functional test and diagnosis is performed offline employing external test equipment [109] [130], recent approaches allow in-field functional testing and diagnosis during NoC operation at system speed [1] [52] [70] [71]. For diagnosis, either all [1] or only particular switches [65] [70] are set to test mode. A switch under test is excluded from normal data communication and only accepts test data. While in case of the former option this implies that NoC operation is preempted completely [1], the latter one allows operation to be continued with a degraded performance [65] [70]. Excluding switches from data communication, however, may temporarily affect connectivity in the NoC. For this reason, in order to maintain full connectivity, additional connections can be added to each switch allowing data packets to bypass a switch under test [65]. Another possibility to continue NoC operation is to perform diagnosis during normal operation of the switch [52] [71]. For the coordination of diagnosis of an entire NoC an additional scheduling method is required that defines the order of switches being diagnosed. The switches can either be diagnosed successively [70] or multiple switches can be diagnosed at the same time [65].

The diagnosis logic can be implemented as dedicated hardware unit [70], as software algorithm [1] [52], or as a combination of both [71]. Diagnosis in software requires the test responses to be communicated to a processing element [1] [52]. In order not to affect the normal data traffic, a switch may implement an additional link to the PE for communication of test responses. Furthermore, the diagnosis logic can be implemented in the NoC either globally as a central diagnosis node [1] [52] or locally at each node [70] [71].

Compared to structural diagnosis, functional diagnosis normally has a shorter diagnosis time resulting from the smaller test pattern set and the possibility to perform diagnosis during NoC operation [23] [27] [50] [135]. However, when diagnosis is performed during NoC operation, the diagnosis time of functional diagnosis approaches depend on the network load [1] [52] [71]. Whereas a high load increases the diagnosis latency because test data and normal data have to share NoC resources [52], in case of approaches using data packets for functional diagnosis a low network load increases the time to gather required test information for diagnosis [1] and observability of faults is reduced [71]. The impact of the network load on the diagnosis time can be reduced by switching components to test mode.

The functional diagnosis approach of Dalirsani et al. [27] is used in this thesis as diagnosis technique of the network layer. With this approach test patterns with a high fault coverage corresponding to the NoC's packet format can be generated. These patterns can be assigned to a switch under test from the software layer, and thus, fit the cross-layer aspect.

### *3.1.3 Protocol-based Diagnosis*

While the fault models of lower layers that consider structural faults or the functionality of components, the fault models used for protocol-based diagnosis of the transport layer are related to communication and comprise the reception of faulty data, delayed data reception, or the complete loss of data. The aim of protocol-based diagnosis is to locate a faulty link or switch on a communication path through the NoC. The localization of permanent faults is based on the analysis of data packets or special test packets and is either performed at NoC start-up [148] or during NoC operation [49] [125].

Packets are communicated on an end-to-end basis passing several links and switches on their way to their destination node. When a packet arrives at its destination node it is checked. The detection of packet corruption or packet loss [125] [148] as well as the detection of packet latencies higher than a tolerated maximum [49] are used as an indication for a possible permanent fault in the NoC. The detection of one of these errors, however, does not necessarily imply that a permanent fault exists in the network because transient faults or the temporary increase of the network load may lead to the same observable errors. For this reason, on the detection of an error, the error first is either stored in a centralized log [125] [148] or in distributed logs [49]. If the same error is reported multiple times in a log, then, with a high probability, this error is caused by a permanent fault in the NoC.

For determining the position of a fault, protocol-based diagnosis approaches associate the observed errors with NoC components. For this purpose, information about the routing used in the NoC is utilized to reconstruct the communication paths of the packets for which an error was detected [49] [125] [148]. In this way, the fault's position is narrowed down to those components that are situated on the respective path. By means of further analyzing the logged information about the components on the path, the component being reported most often is identified to be faulty [49] [125] [148]. This way, protocol-based diagnosis is able to track down a fault's position to a single link or switch [49] [148]. A fault in a link can be further narrowed down to a link channel [103] or even to a single wire [125]. The need to reconstruct the communication paths of packets, however, limits the applicability of protocol-based diagnosis to NoCs with deterministically behaving routing algorithms [49] [148] or routings with only a limited adaptivity [125].

In comparison to structural or functional diagnosis, protocol-based diagnosis does not require additional test structures in switches. However, the code size of the diagnosis algorithm adds to the size of the local memory of a PE (e.g. 2.5 kilobytes [148]). Whereas the time reported in [148] is in the range of mega-cycles, in [125] it is reported that diagnosis is able to pinpoint a fault with only a low effort that requires only 500 packets having passed an 8x8 NoC. Thus, in this case, protocol-based diagnosis is likely to have a smaller diagnosis time than structural and functional diagnosis techniques.

As protocol-based diagnosis is based on the analysis of packets, it mainly covers faults in the data path of links and switches that result in corruption or loss of packets. Nevertheless, also faults in the control path may be observable provided that they lead to corruption or loss as well. The diagnosis capability of protocol-based diagnosis, however, depends on NoC-related aspects such as the underlying routing or network load [49] [125]. A reduced diagnosis capability may even result in links and switches to be erroneously identified as faulty. The occurrence of false positives constitutes the major drawback of protocol-based diagnosis compared to structural diagnosis and functional diagnosis. To minimize the risk of false positives, diagnosis capability can be increased by a careful setup of diagnosis parameters, e.g. threshold value for permanent fault identification, [125] or by minimizing the impact of the network load by prioritizing test packets over normal data packets [49].

In this thesis, a protocol-based diagnosis protocol of the transport layer to locate permanent faults is proposed. In contrast to the diagnosis approaches [49], [125], and [148], the fault localization of the proposed protocol is not based on the analysis of diagnosis logs. Furthermore, it does not use any additional test packets but it reuses those data packets for which the fault has been observed earlier.

### *3.1.4 Cross-Layer Diagnosis*

In contrast to diagnosis techniques on a single network layer, cross-layer diagnosis employs cooperative diagnosis mechanisms on different layers to localize permanent faults. For this purpose, a mechanism provides its available diagnostic information gathered on the respective layer to a mechanism of another layer in a bottom-up or top-down manner. Cross-layer diagnosis is considered as an opportunity to reduce the localization complexity [105] [141] and to reduce the hardware overhead by moving parts of diagnosis to software [18] [24].

Cross-layer diagnosis has been proposed for various kinds of systems such as processors [24], communication networks [98] [105] [129], or general SoC systems [18]. All of these approaches have in common that cross-layer information flow is carried out in a bottom-up manner. The general idea of most of the approaches can be summarized as gathering detailed diagnostic information on lower layers by means of hardware monitoring or sensor units and to provide this information to a diagnosis method in software for fault localization [18] [24] [98]. By this means, diagnosis has access to a wider scope of diagnostic information and thus may be able to locate faults otherwise not locatable [18] [98]. The framework proposed in [105], however, does not consider localization of faults in software. Instead, localization mechanisms are provided on the physical, the data link, and the network layer. If a mechanism is not able to locate the fault it triggers the mechanism on the next higher layer.

Existing NoC-related approaches follow the bottom-up information flow concept as well [1] [52]. In [1], similar to the approaches mentioned above, the information gathered on the network layer is forwarded to a software algorithm for analysis. The diagnosis approach [52] implements hardware counter for diagnosing misrouting or data loss. When such a fault is diagnosed, the information is communicated to a software diagnosis algorithm. There the information together with information gathered on the transport layer is used to locate a permanent fault in the NoC.

In non of the approaches presented in [52] and in [105], implementing diagnosis mechanisms on various layers, cross-layer communication is used for cooperative diagnosis by reusing localization results of one mechanism as input for another one on higher layer. Furthermore, so far, no specific top-down cross-layer diagnosis approaches exist in literature.

This thesis proposes a cross-layer information flow to combine structural, functional, and protocol-based diagnosis techniques. It is composed of a top-down and a bottom-up information flow. While the top-down information flow is used to narrow down the fault's position, the bottom-up information flow is used to provide diagnosis feedback from a lower layer technique to the protocol-based diagnosis.

## 3.2 Fault Tolerance Methods

In literature a variety of fault tolerance methods exists to cope with transient and permanent faults on different network layers [108]. Transient faults vanish after a short time. As a consequence, it is sufficient for a fault tolerance method to correct the effect of the fault by e.g. requesting a retransmission of corrupted data or by restoring the correct state of control logic. In the case of permanent faults, however, the fault's impact has to be tolerated permanently. In that case, retransmitting data or restoring a state does not provide fault tolerance, as the permanent fault would alter the retransmitted data or the state again immediately. In this subsection, first, an overview of methods used to tolerate permanent faults on the different network layers together with their limitations is given. Subsequently, state-of-the-art cross-layer fault tolerance approaches are presented.

### *3.2.1 Single-Layer Fault Tolerance Methods*

In general, a possibility to cope with permanent faults is to mask their effect by means of ECC [81] or modular redundancy [51]. However, the fault remains latent in the NoC and masking may not be possible under certain conditions, e.g. in case of the occurrence of a second fault [108]. For this reason, many fault tolerant approaches that deal with permanent faults either reconfigure the NoC by e.g. replacing defective components by spare components or isolate the faulty components from NoC operation. These approaches make use of the diagnosis result that contains information about the location and the type of a fault.

On ***physical layer***, existing approaches for NoCs add spare wires to the NoC design that are used to replace faulty ones [72] [82]. While a few spare wires per link may be still a tolerable area overhead, the area constraints of NoCs do not allow the implementation of spare for complete links or switches. If the number of defective wires of a link exceeds the number of available spare wires then fault tolerance can no longer be accomplished by replacement but requires higher layer methods.

***Data link layer*** methods consider the links or the crossbar of a switch to be partially faulty. Instead of replacing faulty link wires, available methods for NoCs exclude the faulty wires for data communication and only exploit the remaining fault free ones [76] [102]. For this purpose, these methods implement additional hardware logic to split and reassemble flits and to assign the split flits to the functioning wires. The failure of a crossbar connection or the entire crossbar can e.g. be tolerated by means of a dual crossbar switch architecture [147]. In case of a fault the second crossbar is used for forwarding packets from input to output ports.

Another possibility to tolerate faults in links or crossbar connections is to protect data by means of ECC. Available error correction approaches of the data link layer are targeted to tolerate transient faults [112] [144] [146]. However, in principle, they can also be used to tolerate permanent faults [108]. Data is encoded on sender side and it is checked at each switch on the path towards the receiver. This requires additional encoder and decoder hardware units to be implemented in hardware [95]. Provided that received data is corrupted, it is corrected at a switch if possible. Correction fails if the number of corrupted data bits cannot be corrected by the employed ECC.

If a data link layer method is not able to cope with a faulty link or crossbar, the corresponding component must no longer be used for communication but has to be avoided.

The complete failure of links or switches results in a topology change of the NoC. As a consequence, formerly used communication paths are no longer available resulting in a loss of connectivity between network nodes. Typically, NoC topologies provide an inherent redundancy by offering more than just one path between two communication endpoints. A commonly used methodology on the ***network layer*** taking advantage of these redundant paths to maintain connectivity is fault tolerant routing. A fault tolerant routing may be designed either to adapt to topological changes [41] [48] [74] [111] [122] or to support reconfiguration [4] [44] [68] [110].

Adaptive routings are able to bypass faults by redirecting packets to an alternative path. Their routing decision is based on the fault information locally available at a switch about the fault status of crossbar connections, links, and neighbor switches [41] [48] [74] [111] [122]. To make the fault status of neighbors available at a switch, dedicated signal lines can be utilized [41]. In contrast, the actual representation of a reconfigurable routing behaves deterministically and is not able to adapt to a topology change automatically but the routing has to be recalculated. Routing recalculation on the network layer makes use of a distributed path discovery mechanism where switches broadcast flags in the NoC in order to determine new communication paths [4] [44] [68]. Another possibility is to construct the topology graph centrally, which is then used as an input to a routing calculation algorithm [110].

Besides fault tolerant routing, fault tolerance on the network layer also concerns the arbitration and allocation functionality of a switch. For instance, to prevent grants from being given to wrong input ports of a switch [99], arbitration and allocation stages of a switch can be designed to tolerate faults.

Whereas NoC topologies such as mesh feature redundant paths between two switches, the NIs connecting the PEs to their switch usually are single points of failure. Thus, fault tolerant routing does not provide fault tolerance in case of a faulty NI. Furthermore, methods of the network layer do not deal with the corruption and loss of packets. This has to be provided either on the data link layer or on the transport layer.

Fault tolerance methods on ***transport layer*** comprise the fault tolerant design of NIs [45] [80] as well as error-control methods for packets. The protection of important data in an NI can be achieved by a redundant design of status registers and FIFOs and by means of encoding the data using ECCs [45]. To prevent a PE from being disconnected from the network because of a failure of the NI or of the link connecting the NI and the switch, a PE may be connected to redundant NIs, which in turn are connected to different switches [80].

To deal with the corruption and loss of packets, a commonly used error-control method in NoCs is requesting their retransmission at the sender by means of end-to-end protocols [5] [133]. A packet is checked for faults at the receiver and is positively acknowledged to the sender if it is received correctly. In case of a faulty packet, a negative acknowledgement is sent triggering the retransmission. For detecting packet loss, these protocols make use of a timeout mechanism. If within the timeout period the receipt of a packet was not positively acknowledged packet loss is assumed and the packet is automatically retransmitted by the sender. While these protocols help in case of transient faults, the usage of such protocols as the only measure is not enough to provide fault tolerance for permanent faults because packets being retransmitted pass the faulty component again as they take the same path through the network. Thus, retransmission error-control methods have to be used in combination with a method of a lower network layer, e.g. routing adaptation [5].

Similar to the data link layer, ECCs may be used to tolerate data corruption of packets [125]. In contrast to the data link layer, packets are not check and corrected on a switch-to-switch basis but on an end-to-end basis at the receiver. Thus, the encoder as well as the decoder are implemented as part of the NI [95]. If data cannot be restored, the receiver has to request end-to-end packet retransmission at the sender.

In case of a complete failure of a switch, NI, or PE, the ***software layer*** tasks performed by the respective PE are no longer available as the PE becomes unreachable. This issue is addressed by various online task migration approaches where tasks are assigned to other PEs [39] [60] [90] [115]. Task migration results in a new task mapping. However, this mapping is only valid if the network layer routing can ensure that interdependent tasks can exchange data through the NoC.

### *3.2.2 Cross-Layer Fault Tolerance Methods*

Whereas single layer fault tolerance approaches are able to cope with faults on their own layer, they do not necessarily resolve the fault's impact on another layer. Similar to cross-layer diagnosis, cross-layer fault tolerance is achieved by interaction of methods implemented on different network layers by exchanging information across layer boundaries. Providing information to higher layers enables the relocation of methods from hardware-related layers to the software layer. This reduces the hardware implementation overhead of fault tolerance methods [18] [131].

Most available approaches for NoCs [78] [84] [132] [134] have in common that they, on the one hand, utilize a reconfigurable routing and, on the other hand, make use of software layer methods.

Two cross-layer approaches to calculate the routing in software are presented in [78] and [132]. In [78], in case of a fault, a pre-calculated emergency routing is activated on the network layer. By providing topological information as well as fault information to a software algorithm, an optimal routing is calculated to replace the emergency routing once calculation is finished. In the method for source-based routing proposed in [132], routing paths determined by a hardware discovery mechanism are communicated to a software algorithm. Under consideration of the initial network layer routing the algorithm checks a path if it violates routing restrictions, and if so, VCs are assigned that have to be used on the path in order to ensure deadlock-freedom.

A cross-layer approach for photonic NoCs is presented in [84] to cope with an increasing error rate due to temperature variations. While small variations can be directly handled on the physical layer by a temperature compensation technique, for greater temperature variations this technique is inapplicable. For this purpose, the physical layer technique is combined with two software layer methods that consider temperature information gathered on the physical layer to route around high temperature regions and to migrate tasks for better temperature distribution in the NoC. By combining the methods, the error rate is reduced considerably.

The approach in [134] proposes a layered fault tolerance architecture that provides fault tolerance services on each network layer. The physical layer is used for fault detection only. In case of a fault, it is reported to the data link layer and the network layer methods. Fault recovery always takes place on the lowest possible layer. In case of a faulty link channel, the method on the data link layer redirects a packet to the remaining second channel. If this is not possible, the whole link is considered to be faulty and the network layer method described in [132] is triggered. If an entire network node becomes unavailable, a software layer service migrates the affected tasks to other nodes by assigning them the corresponding object codes, which is stored in an external memory.

Proactive and reactive methods are presented in [9] that by means of interaction increase the dependability of future MPSoC systems with a NoC communication infrastructure. In order to reduce the impact of aging effects that lead to permanent faults, software-controlled thermal management is used to mitigate thermal stress. At the same time, test and diagnosis techniques of lower layers are responsible to detect and locate permanent faults in the hardware. Test and diagnosis results are provided to software methods that use this information to perform online task migration and online reconfiguration in the NoC or for individual processors.

All of the approaches presented above employ two or more methods of different network layers to provide fault tolerance. However, non of the approaches investigates the cross-layer interaction between the individual methods. An overview of the interaction of layered services is given in [134], however, no details are provided about the information being exchanged between the services.

In this thesis, the requirements for exchanging information between methods of different network layers using the standard communication resources of a NoC are discussed and the concepts for two corresponding hardware mechanisms are presented. It is shown that by means of cross-layer interaction fault tolerance for otherwise not fault tolerant deterministic network layer routings can be created. It is further shown that thanks to cross-layer interaction autonomous management of hierarchical units as well as the autonomous routing reconfiguration within the units become possible.

# Chapter 4
# Cross-layer Fault Localization

As permanent faults negatively affect the NoC's functionality or, in worst case, may even result in its complete failure, it is required to exclude affected components from further use. Because of the stationary characteristic of permanent faults, it is possible to localize them, i.e. to identify the faulty NoC component. As discussed in Section 3.1, state-of-the-art diagnosis techniques such as structural diagnosis or functional diagnosis are suitable for a fine-grained localization of permanent faults. However, both techniques have a high localization effort as they require the entire network to be diagnosed. In this chapter, a cross-layer fault localization approach is proposed capable to reduce the localization effort by means of a combination of protocol-based diagnosis of the transport layer with functional diagnosis and structural diagnosis techniques of the lower network layers. In case of the occurrence of a permanent fault in crossbar connections or links, the affected switch is identified by a software protocol and is reported to one of the lower layer techniques. Subsequently, the lower layer technique performs a fine granular diagnosis for the switch to further narrow down the position of the fault. The result of lower layer diagnosis is communicated back to the protocol which uses this information to optimize its own diagnosis process.

The proposed interaction scheme for diagnosis techniques is presented in Section 4.1. The protocol used for fault localization on the transport layer is discussed in Section 4.2. Both lower layer techniques proposed by Dalirsani et al., i.e. functional diagnosis [27] and structural diagnosis [25], are briefly explained in Section 4.3. Section 4.4 evaluates the combinations of techniques regarding their diagnosis quality and the fault localization latency and compares them to the corresponding standalone diagnosis techniques.

## 4.1 Interaction of Diagnosis Techniques

Cross-layer fault localization implies that diagnosis techniques of different layers have to interact with each other. For this reason, diagnosis results have to be communicated across layer boundaries from one technique to another [18] [105]. The proposed interaction scheme used to combine protocol-based diagnosis (P) of the transport layer with functional diagnosis (FD) or structural diagnosis (SD) of lower network layers is shown in Figure 4.1.

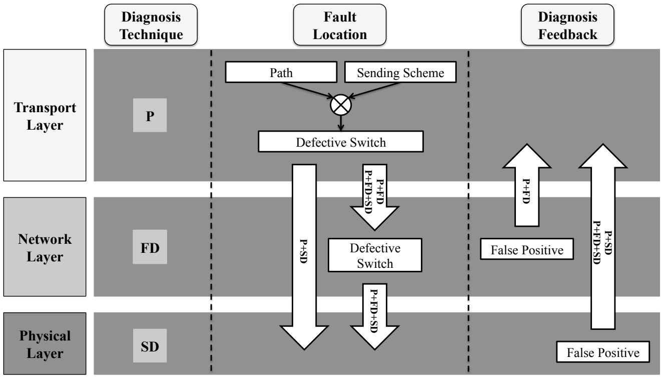


Fig. 4.1: Diagnosis techniques of different layers.

The interaction scheme comprises two information flows: the top-down fault localization and the bottom-up diagnosis feedback. While the top-down flow is used to localize the fault's position within the network, the bottom-up flow provides feedback to the diagnosis protocol P about the diagnosis result of the lower layer techniques. As shown later in Subsection 4.2, operative network components may be diagnosed as faulty (*false positive*) by protocol P due to protocol-related timeouts. When a component reported by protocol P is identified as a false positive by functional diagnosis FD or structural diagnosis SD, this is reported back to P. Listing 1 shows the interaction scheme between the diagnosis techniques as pseudo code.

If a permanent fault is detected in a NoC on an end-to-end basis, first, its position is unknown. For that reason, all the switches $S$ and links $L$ of the network have to be diagnosed to identify the faulty component. The potentially faulty components that have to be diagnosed are represented by set $\Phi$. Out of set $\Phi$, the faulty component is determined by diagnosis function $d_t : \Phi \rightarrow \Phi$ of a technique $t \in \{P, FD, SD\}$. The result of function $d_t$ is the set $\Phi_t$ of components diagnosed as faulty by technique $t$ (line 8, 18, and 25).

The diagnosis process of FD and SD is carried out for all network switches in set $\Phi$, and thus, if used in isolation, FD and SD are performed for the entire network. For both techniques always an entire switch is diagnosed including

**Listing 1** Flow of Diagnosis.

```
$\Phi$: set of potentially faulty links and switches
$\Phi_t$: set of faulty switches diagnosed by technique $t$
$\Delta$: set of all false positives
$\Sigma$: set of all faulty switches reported by P

while true do
    wait for fault detection

    $\Phi := (S, L)$    ▷ initial set $\Phi$ of potentially faulty switches S and links L

    —Top-Down Communication—
    if P then
        $\Phi_P := d_P(\Phi \cap \Phi_{path})$    ▷ where $0 \leq |\Phi_P| \leq 1$
        if combined with lower layer technique then
            $\Phi_P := m_P(\Phi_P)$    ▷ map crossbar or link fault to a switch
        end if
        $\Sigma := \Sigma \cup \Phi_P$    ▷ add component to list
        $\Phi := \Phi_P$    ▷ update set of faulty components
    end if

    if FD then
        $\Phi := \Phi \setminus L$
        $\Phi_{FD} := d_{FD}(\Phi)$    ▷ where $\Phi_{FD} \subseteq \Phi$
        $\Delta := \Phi \setminus \Phi_{FD}$    ▷ determine false positives
        $\Phi := \Phi_{FD}$
    end if

    if SD then
        $\Phi := \Phi \setminus L$
        $\Phi_{SD} := d_{SD}(\Phi)$    ▷ where $\Phi_{SD} \subseteq \Phi$
        $\Delta := \Delta \cup (\Phi \setminus \Phi_{SD})$    ▷ update false positives
        $\Phi := \Phi_{SD}$
    end if

    —Bottom-Up Communication—
    if P then
        if $\Delta \neq \emptyset$ then    ▷ false positive was reported
            adaptation of P
            $\Sigma := \Sigma \setminus \Delta$    ▷ remove false positive from list
        end if
    end if
end while
```

all links $L$ adjacent to a switch, and thus, $\Phi \backslash L$ (line 17 and 24). Protocol P has knowledge about the routing used in the network and can determine the path a packet has taken from its source to its destination. For that reason, the set of components to be diagnosed by P is reduced to those links and switches that are located on the packet's path (line 8). After diagnosis of P has finished, the diagnosis result $\Phi_P$ always contains only one faulty component at maximum. Multiple permanent faults on a path have to be located by consecutive diagnosis runs.

When P is combined with a lower layer technique, the diagnosed faulty component is mapped to the corresponding switch by function $m_P : \Phi_P \rightarrow S$ (line 10), i.e. that switch which contains the faulty crossbar connection or to which the faulty link leads to. The mapping is necessary because FD and SD always diagnose entire switches. The faulty switch is added to set $\Sigma$ of all faulty switches reported by P. Finally, set $\Phi$ is reduced to the one switch identified as faulty and is provided to one of the lower layer technique if available (lines 12 - 13). From input set $\Phi$, diagnosis function of FD and SD determines all the switches with functional or structural faults. All those switches of $\Phi$ not being part of $\Phi_{FD}$ or $\Phi_{SD}$ are fully functional, and thus, belong to the set $\Delta$ of false positives (line 19 - 26).

When a false positives is identified by FD or SD, it is reported back to protocol P as a feedback. In that case, the protocol timeout of P is adapted and the corresponding switch is no longer considered to be faulty (lines 29 - 31).

## 4.2 Protocol-based Fault Localization

Protocol-based diagnosis on the transport layer has the advantage that the required diagnosis effort in terms of time or additionally required diagnosis hardware is less compared to diagnosis on lower network layers. Furthermore, as the underlying network is fully abstracted on the transport layer, diagnosis techniques of that layer are almost independent of the network and its topology. Thus, they can be easily adapted for arbitrary networks.

To localize a fault in the network, the switches and links on the path between a data packet's source and destination have to be diagnosed. To reconstruct the path, the diagnosis technique requires knowledge about the routing used in the network. The reconstruction, however, is only possible if the path can be calculated unambiguously. For instance, in case of fully adaptive routings this is not possible. For that reason, for protocol-based diagnosis on the transport layer routings that behave deterministically as long as the fault is not diagnosed are advantageous.

In the following, an end-to-end protocol is presented that localizes permanent faults in a NoC. It consists of two parts: the base protocol to ensure reliable end-to-end communication in case of transient faults and the diagnosis protocol to locate permanent faults. It utilizes the number of retransmission attempts of a packet to infer the presence of a permanent fault in the network. If for a packet the allowed number of retransmission attempts is exceeded, the diagnosis protocol is activated for that packet and it is forwarded to its destination by a dedicated transmission scheme to locate the fault. The localization granularity of the protocol comprises faulty links and crossbar connections. The failure of a complete switch is determined by the protocol by the failure of all links incident to that switch or by the failure of all its crossbar connections.

### 4.2.1 Base Protocol

Transient and permanent faults can interfere with network communication by causing corruption or loss of data packets. The base protocol implements the *Selective Repeat ARQ* [137] to ensure the reliable end-to-end communication in the presence of faults. When a packet has reached its destination, it is checked for faults and, according to the result, a positive or negative acknowledgement is sent back to the sender of the packet. To temporarily store acknowledgements each node implements a small software queue (*acknowledgement queue*). If this queue is full, no further acknowledgement can be stored and new acknowledgements have to be dropped until the queue has free space again.

If a packet is not received correctly, the packet has to be retransmitted by the sender. The sender stores a copy of each packet in a retransmission buffer. When a negative acknowledgement is received for a packet, it is retransmitted. On receipt of a positive acknowledgement, the corresponding copy is removed from the buffer. However, faults may also lead to packet loss or the loss of an acknowledgement. In both cases, the sender will not receive any acknowledgement and the packet is neither retransmitted nor it is removed from the buffer. This also happens if the acknowledgement is dropped due to a full acknowledgement queue. For this reason, the base protocol implements an adaptable timeout $t_\delta$ to tolerate the loss of packets and acknowledgements. If within this timeout $t_\delta$ no acknowledgement is received for a packet, the packet is considered to be lost and thus is retransmitted by the sender.

Timeout $t_\delta$ depends on static parameters such as the network size. The theoretical minimum timeout $t_{\delta_{min}}$ is defined by the diameter $D(T)$ of the NoC topology $T$ and the latency $\psi$ of switches required to forward a packet from an input port to an output port under zero load. Timeout $t_{\delta_{min}}$ is composed of the time required for the data packet to arrive at its destination plus the time for the acknowledgement to return to the sender [8]:

$$t_{\delta_{min}} = 2 \cdot D(T) \cdot \psi. \tag{4.1}$$

However, it has been shown in [8] and [118] that timeout $t_\delta$ requires a much higher value as the latency $\psi$ at each switch strongly increases compared to the one under zero load because of packets competing for same communication resources. Furthermore, Equation 4.1 does not take into account that the network state changes over time. Such dynamically changing network parameters are e.g. the number of permanent faults, the injection rate, or the network load. For this reason, a static timeout is not suitable but it has to be adapted dynamically to the current network state [118].

### *4.2.2 Protocol Timeout Adaptation*

For the adaptation of timeout $t_\delta$ a distributed adaptation mechanism is used where each network node calculates its own timeout separately. For this purpose, each node measures the roundtrip time, i.e. the required time from sending out a packet until receiving the acknowledgement. The roundtrip time implicitly contains all interferences in the network that result in higher packet latency, e.g. failure of a switch or an increased network load. The network state may temporarily change for a short period of time before returning to its normal state, e.g. because of network traffic with burst character. To avoid considerable fluctuations of timeout $t_\delta$, for the measured roundtrip times, the moving average $t_r$ is calculated. Additionally, the adaptation mechanism provides an interface that allows the timer to be increased by an external source. When protocol P is combined with FD or SD, the external source is the diagnosis feedback. If no external source is used, the adaptation of $t_\delta$ is only based on the moving average $t_r$.

As it will be shown in Subsection 4.2.4, timeout $t_\delta$ is an essential part for the activation of the diagnosis protocol as well as the fault localization process. The diagnosis protocol shall only be activated if the timeouts result from flit loss due to faults, however, not because of timeouts caused by a temporarily high network load. To avoid protocol activations due to load induced timeouts, it is necessary that the value of $t_\delta$ can be quickly adapted to the current load situation in the NoC. Allowing an unlimited growth of timeout $t_\delta$, however, impairs the localization process as $t_\delta$ affects the required localization latency. Thus, maximum value for $t_\delta$ has to be defined.

The function $t : \mathbb{N} \rightarrow \mathbb{N}$ used by the adaptation mechanism to increase and decrease the timeout $t_\delta$ is shown in Equation 4.2.

$$t(i) = t_{init} \cdot (\lceil i_{max}^{i} * \frac{1}{i!} \rceil) \tag{4.2}$$

Starting from the initial value $t_{init}$ where $t_{init} \geq t_{\delta_{min}}$, calculation of timeout $t_\delta$ depends on control parameter $i$. Parameter $i$ is implemented as a simple counter with a maximum value $i_{max}$ $(0 \leq i \leq i_{max})$ to avoid an unlimited growth of $t_\delta$. While for smaller values of $i$ the timeout increases exponentially, for bigger values it increases almost linearly.

The pseudo code of the complete adaptation mechanism is shown in Listing 2. The adaptation process of timeout $t_\delta$ is triggered when an acknowledgement is received or if signaled by the external source. At first, the mechanism calculates the average communication latency $t_r$ (line 5). The value of $t_\delta$ has to be increased if the process is triggered by an external source or if $t_r$ exceeds the current timeout value (line 7). In both cases, control parameter $i$ is increased by one. It is decreased by one if within the time period of $2t_\delta$ the timeout is not further increased and decreasing $i$ does not result in a timeout smaller than the current average latency $t_r$ (lines 9 - 11). For $i$, the new timeout value is calculated according to Equation 4.2 (line 15).

When a packet is sent, as timeout for that packet always the current value of $t_\delta$ is used except in one case. If sending and receiving nodes are direct neighbors, the timeout for the packet is fixed to 1,000 cycles. This is done to achieve a faster localization of a permanent fault by the protocol P in case of packet loss. The rather high value of 1,000 cycles for a distance of one hop was chosen to take high roundtrip times into account caused by high network load.

**Listing 2** Timeout Adaptation Mechanism.

```
$t_r$: round trip time (moving average)
$t_\delta$: timeout
$i$: control parameter ($0 \leq i \leq i_{max}$)

$i := 0; t_\delta := t(i)$
while true do
    wait for external trigger or acknowledgement packet

    $t_r := calcRoundTrip()$

    if external trigger or $t_r > t_\delta$ then
        $i++$
    else if last increase $t_\delta \leq$ now - $2t_\delta$ then
        if $t(i-1) > t_r$ then
            $i--$
        end if
    end if

    $t_\delta := t(i)$
end while
```

## 4.2.3 Localization Principle

The fundamental idea of the protocol's fault localization mechanism is to continuously narrow down the fault position on a path by means of additional test nodes referred to as *intermediate nodes* in the following. Instead of sending a packet directly to its receiver, the packet is sent to the intermediate nodes and is checked there for faults. Checking packets and calculating intermediate nodes is done in software by the processing element of a node. For the calculation, the protocol utilizes information of the transport layer and the network layer as shown in Figure 4.2. On the transport layer no information about the network and its topology is available. However, to select appropriate intermediate nodes, the protocol requires the information which path a packet has used. For reconstructing the packet's path, the localization algorithm requires the following information:

1. the source and the receiver of a communication and
2. the routing algorithm used in the network.

The information about the source and the receiver of a packet is inherently available at every sender. The routing, however, is only available on the net-

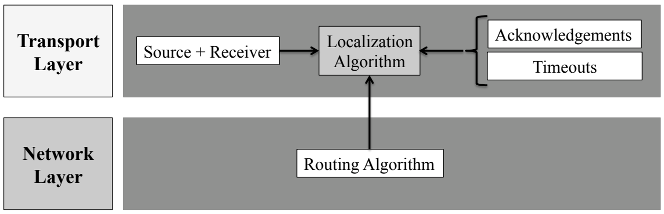


Fig. 4.2: Input parameters of localization algorithm.

work layer. To provide information about the routing to the protocol, the routing algorithm is also implemented in software. By means of both information, the localization algorithm is able to reconstruct the packet's path and to select nodes situated on that path as intermediate nodes. Acknowledgement and timeout information help the localization algorithm to specify the subpath on which the intermediate node has to be located.

## *4.2.4 Diagnosis Protocol*

### 4.2.4.1 Fundamentals

The corruption of packets or their loss can be caused by transient faults as well as permanent faults. On the transport layer, both problems can be detected, but the fault type causing them cannot be identified unambiguously. Diagnosis, however, shall be performed in case of permanent faults only. At sender side, the protocol keeps track of the retransmission attempts of every packet. If a packet cannot be delivered successfully to its receiver after three attempts, the probability is high that this is caused by a permanent fault on path $p = S...R$ between the sender $S$ and the receiver $R$. In this case, the diagnosis protocol is activated for the packet by sender $S$. When the diagnosis protocol is activated for a packet, the packet is no longer directly sent to its receiver $R$, but it is forwarded to intermediate nodes $I$ in order to localize the fault. Every node on path $p$, except $S$ and $R$, may become an intermediate node. The localization logic of the diagnosis protocol is shown in Figure 4.3.

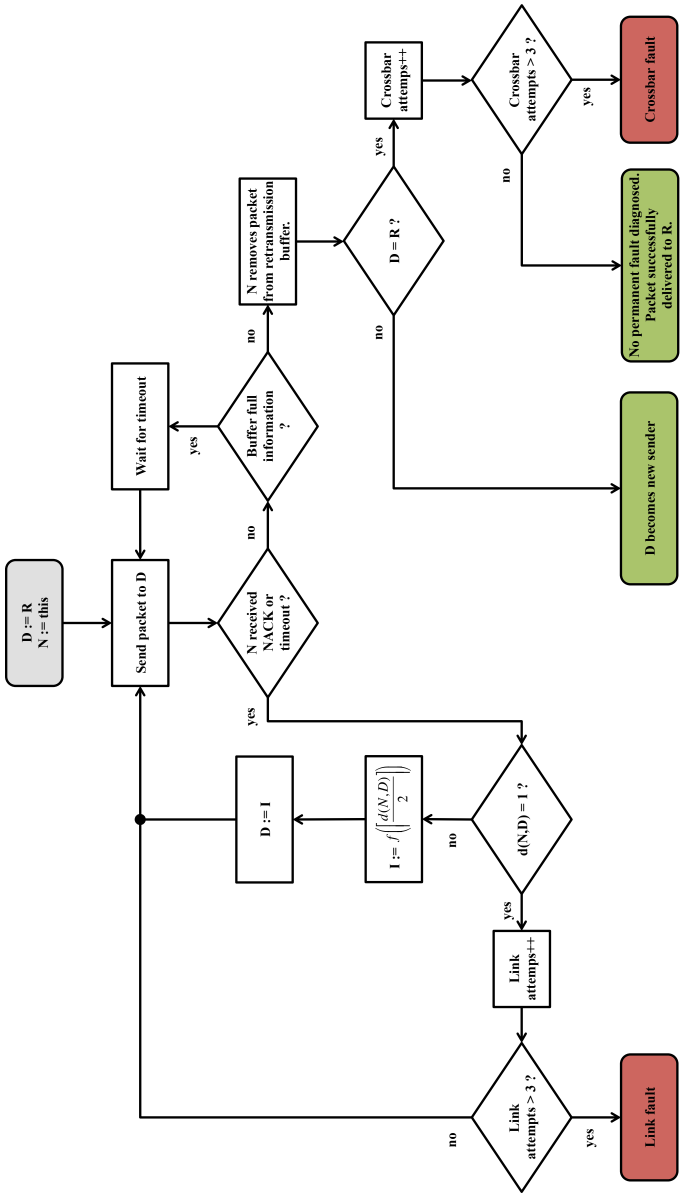

D := R
N := this
Send packet to D
N received NACK or timeout ?
yes
no
Buffer full information ?
yes
Wait for timeout
no
N removes packet from retransmission buffer.
D = R ?
yes
no
Crossbar attempts++
Crossbar attempts > 3 ?
yes
Crossbar fault
no
No permanent fault diagnosed. Packet successfully delivered to R.
D becomes new sender
d(N,D) = 1 ?
no
$I := f\left(\left\lceil \frac{d(N,D)}{2} \right\rceil\right)$
D := I
yes
Link attempts++
Link attempts > 3 ?
yes
Link fault
no

Fig. 4.3: Localization logic of the diagnosis protocol.

#### 4.2.4.2 Localization Logic

The localization logic of the protocol is able to pinpoint a permanent fault to a crossbar connection or to a link on path $p$. For the localization, intermediate nodes are selected as new destination $D$ used to check the packet. From all network nodes on path $p$, $S$ selects the node as intermediate node $I$ that is situated at half distance from $S$ to $R$ and forwards the packet to $I$. On receipt of the packet, $I$ checks it for faults. If the packet is correctly received, this implies, that the first half of path $p$, i.e. subpath $p_1 = S...I$, between $S$ and $I$ is fault-free and the fault has to be either located on the second half $p_2 = I...R$ or on the crossbar connection of $I$ that connects both subpaths. In this case, $I$ sends a positive acknowledgement (ACK) to $S$, stores the packet in its retransmission buffer, and becomes the new sender $N$ of the packet. When $S$ receives the positive acknowledgement, it removes the packet from its retransmission buffer. If $I$ sends a negative acknowledgement (NACK) to $S$ or in case of timeout $t_\delta$, the fault is located on subpath $p_1$ and $S$ remains the sender of the packet. In the following, fault localization is continued for the affected subpath only.

To narrow down the fault's location on a subpath, the current sender $N$ of the packet further bisects this subpath, e.g. $p_{1,1}$ and $p_{1,2}$, by selecting the next intermediate node as destination $D$. The bisection process is controlled by the responses from $D$ and it is repeated as long as $N$ either receives a NACK from the corresponding destination $D$ or a timeout occurs in case of no response. If the distance between $N$ and $D$ becomes one hop ($d(N,D) = 1$), the fault has to be located on the link between $N$ and $D$. To rule out that the link fault was caused by a transient fault, the packet is retransmitted for three more times from $N$ to $D$. After three failed retransmission attempts, the link is identified as faulty by the localization logic.

The checking of packets is done at the processing element of an intermediate node. This implies, however, that at an intermediate node the packet does not traverse the switch's crossbar connection $i \rightarrow o$ that connects the two subpaths $N...I$ and $I...D$. The reason for this is that the packet is forwarded from the switch's input port $i$ to the local port $L$ using crossbar connection $i \rightarrow L$. Later on, the packet is reinjected using the crossbar connection $L \rightarrow o$. As a consequence, a potential permanent fault on $i \rightarrow o$ is not identified by the protocol as it is bridged.

To be able to identify a faulty crossbar connection, an intermediate node $I_a$ requires the information that the fault is located neither on subpath $S...I_a$ nor on the subpath $I_a...R$. As the packet was successfully sent to $I_a$, the fault cannot be located on subpath $S...I_a$. To obtain the information about the subpath

$I_a...R$, intermediate node $I_a$ first sends the packet to receiver $R$ before further bisecting this subpath. If $I_a$ would immediately start the bisection process for subpath $I_a...R$ by choosing intermediate node $I_b$ as destination, the fault state of crossbar connection $i \to o$ could not be determined unambiguously in case of an ACK from $I_b$. The reason for this is, although the ACK confirms that the permanent fault is not located on subpath $I_a...I_b$, the fault still may be located on the not checked subpath $I_b...R$. If $I_a$, however, sends the packet first to $R$, in case of an ACK, this confirms that the whole path from $I_a$ to $R$ is fault-free. With this information $I_a$ is able to conclude an internal crossbar fault. The conclusion is based on the following three statements:

1. Because the diagnosis protocol is activated for path $p$, a permanent fault exists on that path with a high probability.
2. The node has become intermediate node $I_a$ and has successfully received the packet. Thus, the whole subpath $S...I_a$ is fault-free.
3. The packet could be forwarded successfully to $R$. Thus, the subpath $I_a...R$ is also fault-free.

When $I_a$ determines a possible internal crossbar fault, a counter for the corresponding connection $i \to o$ is increased. If the same crossbar connection is reported to be faulty more than three times, it is assumed to be faulty.

Retransmission buffers can only store a limit number of packets, and thus, when the buffer is full, an intermediate node $I$ cannot accept arriving packets but has to drop them. If the distance between the sending node and $I$ is one hop, dropping packets may lead to a working link being diagnosed as faulty by the protocol if the packet cannot be accepted three times. To prevent this, $I$ sends a *buffer full* information back to the sending node signaling that no fault has occurred. The sender retransmits the packet to $I$ after timeout $t_\delta$ without increasing the number of retransmission attempts.

When diagnosis process of the protocol is finished, a fault is either classified as link fault or as crossbar fault. If protocol P is combined with one of the other techniques, the diagnosed fault is mapped to a particular switch (cf. Listing 1 line 10). In case of a link fault between node $N$ and its neighbor $D$, $D$ is reported as the faulty switch, i.e. $m_P(\Phi_P) = D$. In case of a crossbar fault, node $N$ that has diagnosed the fault is reported, i.e. $m_P(\Phi_P) = N$.

### *4.2.5 Evaluation*

In this subsection, the diagnosis capability of protocol P and its implementation costs are evaluated. The diagnosis capability is evaluated for an 8x8 mesh NoC using uniform random traffic and different data injection rates. Table 4.1 summarizes the setup of the simulation model.

Table 4.1: Simulation Model Setup

| Parameter | Setup |
|---|---|
| Topology | 8x8 Mesh |
| Routing | XY |
| Arbitration | Round Robin |
| Switching | Wormhole |
| Packet Size | 5 Flits |
| Traffic Pattern | Uniform Random |
| Data Injection Rate | Moderate: 0.14 flits/cycle/node<br>High: 0.28 flits/cycle/node |
| Simulated Time | 500,000 cycles |

#### 4.2.5.1 Diagnosis Capability

The occurrence of transient faults can activate the diagnosis protocol. For instance, this may happen if a packet becomes corrupted by transient faults each time it is retransmitted from sender to receiver. In a first evaluation, the probability of occurrence is studied at which transient faults are erroneously identified as permanent faults. For this purpose, transient faults are injected into the NoC at random locations for different probabilities. The measured number of erroneously identified permanent faults for moderate and high network load are shown in Table 4.2. The results show that only in case of unrealistic high probabilities per cycle (cf. Subsection 2.3.1), transient faults are reported as permanent ones by the diagnosis protocol in both load situations. While for moderate load this happens for a probability of 70% per cycle or higher, for high load this can already be observed for a probability of 30% per cycle. The reason why permanent faults are reported at lower probabilities for high load can be attributed to the higher number of flits in the NoC. When a transient

Table 4.2: Number of Erroneously Identified Permanent Faults

| Probability per cycle | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|
| Moderate load | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 13 | 35 | 60 |
| High load | 0 | 0 | 2 | 7 | 19 | 42 | 82 | 151 | 253 | 416 |

fault occurs, the probability that a flit is affected by the fault is higher than in case of moderate load.

In a second evaluation permanent link and crossbar faults are injected into the NoC. For different timeouts $t_\delta$ it is investigated whether P can correctly locate the faults or if functioning components are diagnosed as faulty (false positives). For this purpose, five different fault sets with each containing five different permanent faults are used. The position of each fault and its time of occurrence are randomly chosen. During simulation $t_\delta$ can be increased two times, i.e. $i_{max} = 3$ (cf. Equation 4.2). When a fault is diagnosed, the corresponding component is shut down. In order to bypass shut down components, software-based packet rerouting is used, which is described later in Chapter 5. The average number of false positives for different timeouts $t_\delta$ and different network loads is shown in Figure 4.4.

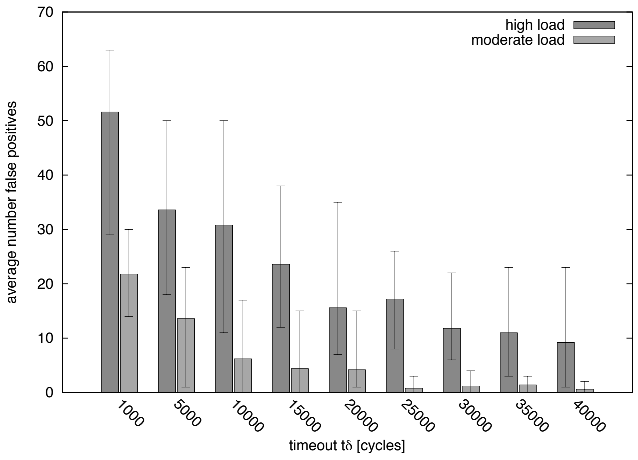


Fig. 4.4: Average number of false positives.

The results show that the diagnosis capability of P strongly depends on timeout $t_\delta$. In principle, the protocol was able to find all the faults defined in the fault sets for all timeouts. It can be observed, however, that for small timeouts a high number of false positives is diagnosed. The reason for diagnosing false positives originates from a high load in the NoC. In the case of faults, the load increases because of the affected communication resources become unavailable. For small values of $t_\delta$, the load increases even more because of the more frequent packet retransmissions resulting in congestion of the NoC. If an acknowledgement cannot be injected immediately, it has to be stored in the acknowledgement queue of the receiver of a packet. In case of congestion, this queue eventually reaches its limit and new acknowledgements have to be dropped. As the absence of acknowledgements is considered by the protocol as a loss due to a fault, the dropping can eventually activate the diagnosis protocol, and thus, can result in false positives being diagnosed. For higher values for $t_\delta$, congestion in the NoC is avoided and only in few cases acknowledgements have to be dropped. Thus, with the increase of the timeout, the number of false positives decreases.

As indicated by the minimum and maximum values in Figure 4.4, the number of false positives varies for the different fault sets. For instance, while for moderate load in case of one fault set no fault positives were reported for $t_\delta = 10,000$ cycles, for another set 17 false positives were diagnosed by P. For timeouts $t_\delta > 20,000$ cycles, only approximately one false positive in average is diagnosed. In case of high load, however, the measured minimum does not fall below an average of approximately nine false positives ($t_\delta = 40,000$ cycles) for all considered timeout values.

#### 4.2.5.2 Implementation Costs

The total code size of protocol P is approximately 28.4 KiB and is composed of the code sizes of the base protocol (cf. Subsection 4.2.1) and the diagnosis protocol (cf. Subsection 4.2.4). Compared to the base protocol, the code size of the diagnosis protocol (3.3 KiB) is small and only corresponds to 11.6% of the total size (cf. Table 4.3).

To store packets for retransmission at a sender, additionally a retransmission buffer is required. In this work, a retransmission buffer can store up to 10 packets. Each packet has a size of 160 bits, and thus, the buffer size is 200 bytes per network node.

Table 4.3: Protocol Code Size [KiB]

| Base Protocol | Diagnosis Protocol | Total |
|---|---|---|
| 25.1 | 3.3 | 28.4 |

## 4.3 Combination of Diagnosis Techniques

For the combination with software protocol P, functional diagnosis (FD) [27] and structural diagnosis (SD) [25] approaches proposed by Dalirsani et al. are used. FD on the network layer and SD on the physical layer are both single layer techniques used to diagnose single switches. Within the context of this thesis, SD and FD are used to diagnose the entire NoC. While SD technique can be applied to all network switches in parallel, not all switches can be diagnosed at the same time in case of FD. For this reason, a schedule for FD has been proposed that allows a maximum number of switches to be diagnosed in parallel [121]. The proposed schedule for FD is presented in Subsection 4.3.1.

### *4.3.1 Functional Diagnosis*

On occurrence of a structural fault, functional diagnosis (FD) [27] determines the affected functionality of a NoC switch. The supported functional failure classes of FD are:

1. *Data corruption*: packet payload is altered.
2. *Misrouting*: packet is forwarded to a wrong switch output.
3. *Packet loss*: loss of a whole packet.
4. *Flit loss*: loss of one or more flits of a packet.
5. *Packet duplication*: duplication of a whole packet.
6. *Flit duplication*: duplication of at least one flit of a packet.

All structural faults that do not cause one of these functional failures belong to the *residue class*. If FD diagnoses a fault that is associated with the residue class, the whole switch has to be assumed as faulty.

The test patterns utilized by FD correspond to a valid packet format and are referred to as test packets. Test packets are generated offline by means of a satisfiability solver (SAT) based method. Using test packets allows the reuse of the communication structure of a NoC as test access mechanism (TAM). During the diagnosis process, all the neighbors of a switch-under-diagnosis (SUD) send test packets to the SUD's input ports. Test packets for the local input are sent by the SUD's processing element. While being diagnosed, the SUD has to remain idle and is not available for data communication. All neighbors sending test packets to the SUD cannot be diagnosed in parallel, instead, they remain operative and can accept and forward data packets.

When FD is used as standalone technique, set $\Phi$ (cf. Listing 1) contains all switches, and thus, the entire NoC has to be diagnosed. The utilized diagnosis schedule with a maximum of switches being diagnosed in parallel is shown for an 8x8 NoC in Figure 4.5. The schedule consists of five diagnosis itera-

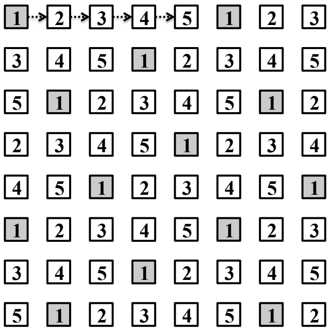


Fig. 4.5: Diagnosis schedule FD.

tions in total. The number shown for each switch represents the iteration the switch is diagnosed in. For larger mesh networks the same number of iterations is required. If FD is combined with protocol P, referred to as P+FD, FD is performed only for the reduced set $\Phi$ containing one switch. In this case, the schedule is not required.

When FD has finished, all faulty nodes from set $\Phi$ are identified. The remaining switches are considered as fully functional (set $\Delta$), and thus, are false positives.

### *4.3.2 Structural Diagnosis*

For diagnosis of structural faults in the logic of a switch, structural diagnosis (SD) [25] based on scan design is used. The fine diagnosis granularity allows the localization of faulty gates and wires. As TAM each switch implements additional scan chains. During diagnosis test patterns are applied to these scan chains.

If SD is used as standalone technique, from the set of all network switches $\Phi$, the faulty switches have to be identified first. For that purpose, the test pattern set is applied to all the network switches in parallel and the test responses of neighbored switches are compared. If the responses of a switch differ from the others, the switch has to be diagnosed. In a second iteration the same test patterns are applied to the faulty switches. The test responses are analyzed at the diagnosis service point and the faulty gates and wires are determined. For the combination of SD with P (P+SD), like in case of FD, diagnosis is performed for the reported switch only. For this reason, identification of faulty switches, as required for standalone SD, can be omitted.

During the identification of faulty switches as well as during diagnosis, a switch is not available for data communication. If no faults are found, the switch is classified as false positive.

## 4.4 Evaluation

The combinations of diagnosis techniques are evaluated with regard to the diagnosis quality and the fault localization latency. The results are compared to those of the corresponding standalone techniques. For evaluating the localization latency, the same simulation setup as in Subsection 4.2.5.1 is used.

### *4.4.1 Definitions*

***Diagnosis quality***: The quality of a diagnosis technique states whether a fault can be localized and how exact its position can be determined. The criteria used to evaluate the quality of diagnosis techniques are:

1. *Fault coverage*: indicates the percentage of structural faults that can be detected by a diagnosis technique [136].
2. *Localization accuracy*: defines the granularity in which a fault can be localized.
3. *Number of false positives*: states the number of fully functional components erroneously being diagnosed as faulty by a diagnosis technique.

The diagnosis techniques P, FD, and SD are based on different fault models. While P diagnoses faulty links and crossbar connections, FD determines functional faults of a switch. SD, however, localizes faulty gates and wires. For this reason, the direct comparison of the localization granularity of techniques is not reasonable. In the following, the localization accuracy of a technique is evaluated on the basis of whether the covered structural faults can be correctly diagnosed with respect to the technique's fault model.

***Localization latency***: The localization latency defines the total number of cycles required from the detection of a permanent fault until it is localized by a diagnosis technique.

### *4.4.2 Diagnosis Quality*

In this section, at first, the quality of standalone techniques P, FD, and SD is discussed in Subsection 4.4.2.1. The quality results for the combinations P+FD, P+SD, and combination of all three diagnosis techniques (P+FD+SD) are presented in Subsection 4.4.2.2.

#### **4.4.2.1 Standalone Diagnosis Techniques**

The diagnosis process of protocol P is based on the analysis of packets. Thus, only those permanent faults can be diagnosed that either lead to a corruption of packets or to a complete packet loss. For instance, a permanent fault resulting in misrouting of a packet cannot be detected by P, if the packet eventually arrives at the receiver within a tolerated time period. For that reason, P does not cover all possible structural faults. All faults covered by P are either mapped to a link or a crossbar connection. For example, a faulty input buffer slot is diagnosed as link fault. Thus, some of the faults cannot be localized accurately. Furthermore, the evaluation of protocol P (cf. Section 4.2.5.1) shows that, depending on the network load, false positives may be reported. While for moderate network load less than one false positive is diagnosed on average, for high network load, false positives are reported for all considered timeout values. Because of its dependence on the network load, in the following, the quality of P is evaluated for moderate load ($P_{mod}$) and for high load ($P_{high}$).

For functional diagnosis FD and for structural diagnosis SD it is possible to generate test patterns that cover almost 100% of all possible structural faults. In contrast to P, the localization accuracy does not depend on the network load. Evaluation in [27] shows that around 80% of structural faults can be mapped to one of the functional failures defined by the six classes (cf. Section 4.3.1). For all other faults the entire switch has to be assumed to be faulty. Because of SD's fine diagnosis granularity, the gate or wire affected by a structural fault can be identified. For both techniques no false positives can occur if valid patterns are used.

Because of the protocol's smaller coverage and because of false positives, diagnosis quality for both load situations is less than the one of FD and SD. FD's quality is less than the one of SD because in 20% of faults the complete switch has to be considered as faulty. According to their diagnosis quality this results in the following order of diagnosis techniques:

$$P_{high} < P_{mod} < FD < SD \tag{4.3}$$

### 4.4.2.2 Combined Techniques

The considered combined techniques are P+FD, P+SD, and P+FD+SD. For each of these combinations, FD or SD is triggered when P diagnoses a fault. This implies that the coverage of the combined techniques is defined by protocol P. If P does not cover a fault, the combined technique does not cover it as well. For that reason, the coverage of all P+ techniques is less than the one of FD and SD. The localization granularity of a combined technique is defined by the technique utilized on lowest layer. As in the case of the corresponding standalone techniques, the accuracy of P+FD is less than the one of P+SD and P+FD+SD. It is still possible that P reports false positives, however, FD and SD techniques are able to identify false positives due to their fine-grained diagnosis and to provide feedback to P. The feedback is used to adapt its timeout value and to reduce the number of diagnosed false positives. While the quality of P+SD and P+FD+SD is the same, the quality of P+FD is less due its reduced accuracy. Thus, the order on combined techniques regarding their quality is as follows:

$$P + FD < P + SD = P + FD + SD \tag{4.4}$$

Compared to P, all P+ techniques generally have a higher diagnosis quality because of the higher localization accuracy provided by the lower layer technique. In best case $P_{mod}$ is comparable to P+FD regarding its quality. This is the case when $P_{mod}$ does not diagnose false positives. However, in average case, false positives are diagnosed, and thus, the quality level of $P_{mod}$ is lower than the one of P+FD. Taking into account the order of standalone techniques (cf. Equation 4.3) and of combined techniques (cf. Equation 4.4), results in six different quality levels. The assignment of diagnosis techniques to quality levels is shown in Table 4.4, where *level 1* represents the lowest diagnosis quality and *level 6 the highest one*.

Table 4.4: Quality Levels of Techniques

| Quality level | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| $P_{high}$ | $P_{mod}$ | P+FD | P+SD<br>P+FD+SD | FD | SD |

## 4.4.3 Fault Localization Latency

For investigating the fault localization latency of the different diagnosis techniques, the NoC is simulated for 20,000 cycles without faults. Then a single permanent link or crossbar fault is randomly injected and the time between the detection of the fault and its localization is measured. The measured average localization latency for the diagnosis techniques is shown in Table 4.5.

The results of standalone techniques reflect the diagnosis effort of each technique. For FD and SD the pattern set consists of a few hundreds of test patterns. In case of FD, 32,500 cycles are required to apply all patterns to every switch in the network using the diagnosis schedule (cf. Subsection 4.3.1). For SD, the total time for identification phase and actual diagnosis phase is 99,754 cycles. Compared to these two techniques, the effort of P is much lower. On the one hand, diagnosis is only performed for the switches on a path and, on

Table 4.5: Average Localization Latency [cycles]

| Diagnosis technique | moderate load | high load |
|---|---|---|
| P | 315 | 471 |
| FD | 32,500 | |
| SD | 99,754 | |
| P+FD | 6,751 | 6,922 |
| P+SD | 50,128 | 50,348 |
| P+FD+SD | 56,628 | 56,799 |

the other hand, during diagnosis only one packet is used to locate the fault. While localization latency of FD and SD are independent of the network load, for P the time increases slightly for high network load. The reason for this is that latency of acknowledgement packets used during diagnosis increases.

The combination of FD or SD with software protocol P results in a faster localization of faults compared to the standalone techniques. Due to the localization of the faulty switch by protocol P, set $\Phi$ of faulty switches is reduced to one switch. Thus, full network diagnosis by FD or SD is not required and diagnosis scheduling (for FD) and identification phase (for SD) are omitted. Compared to FD, the localization latency of P+FD is reduced by approximately 79% and corresponds to the time required for one iteration of the diagnosis schedule. As the identification phase is omitted, the localization latency of P+SD is only 50% compared to SD. When all three techniques are combined, the localization latency is comprised of the time required by P to locate the faulty switch, the time for one iteration of the diagnosis schedule of FD, and the time required for the diagnosis phase of SD. As P depends on the network load, the time of all P+ techniques increase slightly for high network load.

## 4.5 Summary

The results show that cross-layer fault localization by means of the combination of the diagnosis protocol of the transport layer with functional diagnosis or structural diagnosis of lower network layers show a good tradeoff between diagnosis quality and diagnosis performance compared to the standalone techniques. The combinations enable functional diagnosis and structural diagnosis to provide feedback to the protocol which is used to improve the protocol's diagnosis capability resulting in a higher diagnosis quality compared to protocol-based diagnosis used in isolation. Compared to functional diagnosis and structural diagnosis, the quality, however, is reduced due to the smaller coverage of the protocol. While the fault localization latency of the combinations is increased compared to the protocol, it is considerably smaller than the ones of the lower layer techniques. The reason for this is that the faulty switch is identified by the protocol, and thus, the necessity to perform diagnosis in the entire network is omitted. Results show a localization speedup of about factor five compared to functional diagnosis and factor two compared to structural diagnosis.

# Chapter 5
# Cross-Layer Fault Tolerance

In contrast to single layer approaches, cross-layer fault tolerance approaches derive their benefit from incorporating information about the network and faults from different layers. This allows methods of different layers to deal with faults cooperatively in order to provide fault tolerance. By providing network information to the software layer, cross-layer fault tolerance enables the handling of lower layer faults to be performed by a software method omitting the need to implement the respective functionality in hardware. Information of lower layers is communicated to the software layer in a bottom-up manner where the information has to cross one or more layers, as shown in Figure 5.1. To provide information to the next higher layer, the information has to be con-

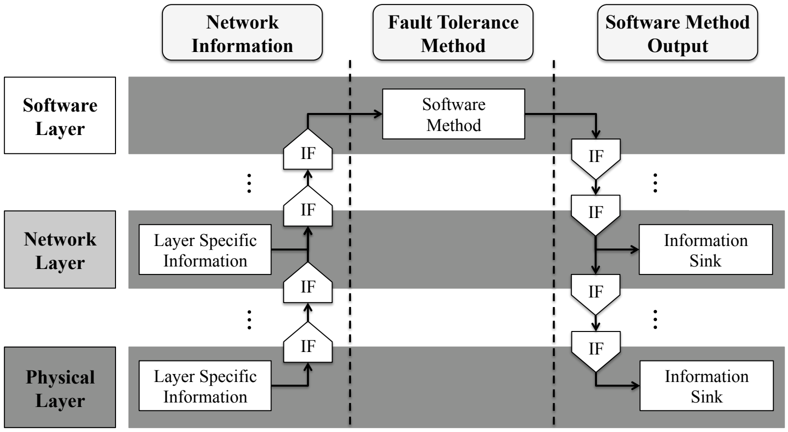


Fig. 5.1: Information flow to and originating from the software layer.

verted into an information unit processible by the next higher layer, e.g. an analog value on the physical layer has to be converted into a digital signal value on the data link layer and vice versa. For this purpose, layers must offer each other an interface (IF). The output information of a software fault tolerance method has to be communicated in a top-down manner to the respective layers where the information is used in order to provide fault tolerance.

For providing information to the software layer and back to lower layers the methods presented in this chapter make use of the NoC resources and interfaces of standard data communication. As these resources and interfaces are not exclusively used for exchanging network information across layers, it has to be ensured that cross-layer communication is not blocked by normal data traffic.

In this chapter, first, two methods are discussed that, on the one hand, allow link state information to be exchanged between data link layer and higher layers and, on the other hand, prevent communication resources from being blocked by data flits. In the following, both methods are subsumed as communication resource management method (CRM) (cf. Subsection 5.1). The required information as well as the respective information flow is shown for each of the methods. Subsequently, *Software-based Packet Rerouting* (SBR) and *Reconfigurable Hierarchical Routing* (RHR) methods are presented (cf. Subsection 5.2 and Subsection 5.3), that are able, by incorporating information of different network layers, to redirect packets and to adapt network layer routing, respectively, on the software layer. An overview of the three methods presented in this chapter is given in Table 5.1.

Table 5.1: Fault Tolerance Methods

| | CRM | SBR | RHR |
|---|---|---|---|
| Supported Fault Type | transient & permanent | permanent | permanent |
| Implementation Layer | data link | software | network & software |
| Fault Model | crossbar & link failure | crossbar, link, & switch failure | link & switch failure |

## 5.1 Management of Communication Resources

In case of a failure of a communication resource, it becomes unavailable for communication. The information about the availability of a resource is essential for fault tolerance methods. In this chapter, a method is presented that enables the exchange of status information between data link layer and higher layers (cf. Subsection 5.1.1). Furthermore, the failure of a resource may have an impact on the data flow in the NoC by blocking communication. For this purpose, a method for data flow management is proposed in this chapter that prevents resources from being blocked by releasing reserved resources in case of flit loss (cf. Subsection 5.1.2).

### 5.1.1 Availability Status Communication

In order to react to the failure of a communication resource, a fault tolerance method has to have knowledge about its availability. The resource's availability is captured by its *availability status*. This status is determined by fault diagnosis and it is utilized by fault tolerance methods. Thus, it is required that the status is communicated to the fault tolerance methods.

On the data link layer, the availability status can be communicated by using dedicated hardware signals. This is reasonable if the status is only used by local switch mechanisms or by mechanisms implemented in neighboring switches. However, if the status is required globally in the NoC, the usage of hardware signals would result in a huge wiring overhead. On the network layer the availability status can be communicated as flits using the available NoC communication infrastructure. Utilizing the NoC communication infrastructure allows the existing NIs to be used to provide the status to the software layer. To enable the status to be communicated between data link layer and network layer it is required that the status information is converted from flit payload to signal states and vice versa. The status exchange between network and data link layer is shown in Figure 5.2.

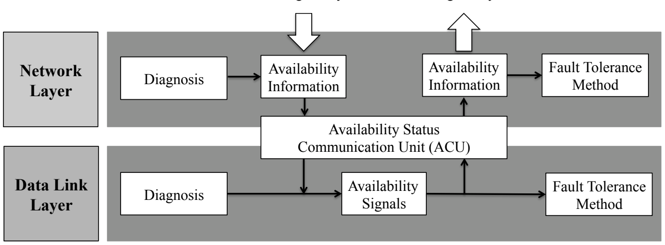


Fig. 5.2: Communication of availability status across network layers.

The proposed method in this thesis enables the communication of the availability status of link channels and crossbar connections. An example for the west port of a switch is shown in Figure 5.3. The availability of a switch is represented by the availability status of all incident links, i.e. a switch is unavailable if all incident links are unavailable. The availability status of each communication resource is represented by a one bit hardware register. To notify a neighbor switch about the availability of a link channel, an *availability signal* is used on the data link layer. The availability signal is reset if either the channel becomes unavailable or if all crossbar connections originating from the corresponding input port of a switch are unavailable.

To allow the exchange of the availability status between data link layer and higher layers, a switch implements the addressable *Availability Communication Unit* (ACU). It implements one interface for status setting (SET) and status communication (COM) to higher layers. From the point of view of the switch's router, the ACU is considered as an additional communication endpoint to which flits have to be forwarded to and which injects flits.

SET and COM interfaces enable diagnosis or fault tolerance methods to be performed on higher layers. If a fault has been diagnosed, the diagnosis result is communicated by means of a flit to the ACU where the availability status is set accordingly. The COM interface to provide status information to higher layers has been designed and implemented in the master thesis [2]. The ACU monitors the availability status of all link channels and crossbar connections of a switch and in case a status is set to 'unavailable', a flit is generated with the corresponding information. In SBR the flit is sent to the local PE. In RHR it is send to all network nodes within a hierarchical unit.

If a communication resource becomes unavailable, in this thesis, it is assumed that this resource is shut down.

### *5.1.2 Data Flow Management*

Switching and flow control are essential mechanisms for enabling data flow in the NoC. Common NoC switching methods make use of a special setup flit or the head flit of a packet to reserve an output port at each switch the flit passes. An output port is exclusively reserved for an input port, and thus, flits from other input ports may not use this output port at the same time. The tail flit of a packet is used to cancel the reservation. Flow control allocates buffer resources to flits and stops a switch from sending further flits if a buffer cannot

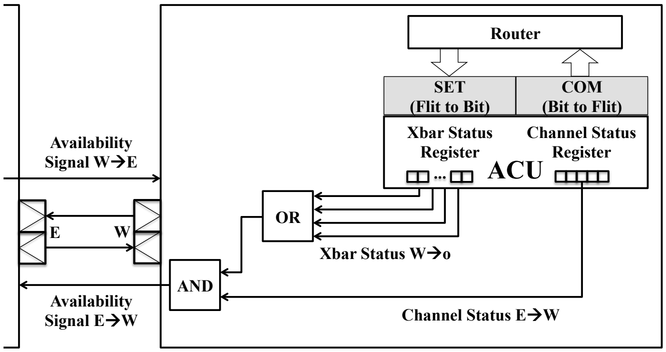


Fig. 5.3: Architecture of *Availability Communication Unit*.

store them at the moment. The loss of head and tail flits has a negative impact on both mechanisms.

The loss of a tail flit prevents output reservations to be canceled, and thus, all flits at other input ports requiring this output port are blocked. Depending on the used switching method a head is forwarded without waiting for the tail to arrive at the same switch. Thus, output reservations at several switches made by the head flit are not canceled. This situation also occurs if a link or crossbar connection is shut down while being used for communicating a packet. That is because the tail flit can only cancel reservations up to the shut down resource. All reservations beyond this point are not canceled as the tail flit cannot further be forwarded. If a head flit is lost, body and tail flits block a buffer as they cannot be routed. When the buffer is full, flow control does not accept any new flit to be stored in the buffer.

Uncanceled reservations and blocked buffers cause a backlog of flits in the NoC resulting in a blockage of the complete NoC eventually. In turn, the blockage prevents fault tolerance methods on higher layers such as adaptive routings to take effect. Thus, it is mandatory to dissolve the blockage by canceling reservations and by dropping not routable flits.

A *Data Flow Management* (DFM) method that prevents buffers from being blocked and that cancels reservations is presented in the master thesis [32]. It is designed for input-buffered NoC switches with wormhole switching and credit-based flow control. While the focus of the master thesis is on transient faults, the proposed error recovery mechanisms can also be used in case of permanent faults. The DFM is implemented as an additional hardware unit in each NoC switch. As input information the DFM employs information about the type of flits to reset grants given by the arbiter and to control write access of input buffers, as shown in Figure 5.4.

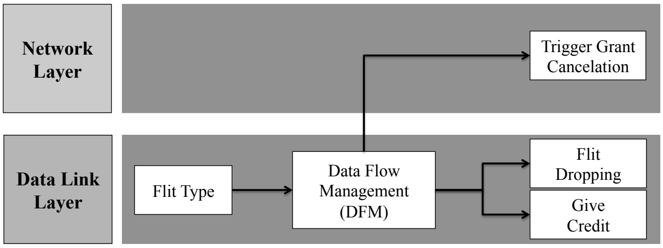


Fig. 5.4: Information flow of *Data Flow Management*.

The proposed method keeps track of the type of the incoming flits at an input port of a switch and checks that the head and tail flit of a packet are received. Tail flit loss of a packet is considered if before receiving the packet's tail flit either the head flit of the next packet arrives or if no further flit is available in the input buffer. The latter guarantees that all reservations made by the head flit are canceled even though no further packet is sent through the NoC on the same path. If a tail is lost, the method simply resets the grant given by the arbiter to cancel the output reservation of the corresponding input port.

Head loss is considered by the method if the flit following a tail flit is not a valid head. In case of a head loss, all incoming flits at the corresponding input port are dropped before they are stored in the buffer. This prevents buffers to be blocked by not routable flits. To prevent a mismatch between the number of available buffer slots and the credit counter of the sending switch, a credit is passed back for each dropped flit. Dropping stops when the next head flit arrives at the input port.

## 5.2 Software-Based Packet Rerouting

In case of failure and shutdown of a crossbar connection, a link, or a switch, the component becomes unavailable for data communication. This implies that all communication paths in the NoC passing this component are no longer valid. Topologies such as mesh or torus have an inherent redundancy offering several redundant paths between each source and destination pair. In general, this allows packets to bypass a shut down component by using an alternative path. Deterministic and oblivious routings implemented on the network layer, however, cannot exploit alternative paths, and thus, as a consequence of component shutdown, connectivity in the NoC is no longer provided between network nodes.

Providing network information to the software layer offers a solution to this problem as it enables the rerouting of packets in software. Software-based rerouting allows packets to use alternative paths which cannot be exploited by the network layer routing. For the calculation of an alternative path knowledge about:

- the routing used on the network layer,
- the NoC's topology, and
- the availability of the communication resources

is required. In this section, *Software-based Packet Rerouting* method for NoCs is presented to tolerate the failure of crossbar connections, the failure of a link channel or the complete link, as well as switch failure. As rerouting is performed in software, no additional hardware to calculate alternative paths is required in a switch. It is assumed that packets that cannot be forwarded to an output port due to an unavailable communication component are forwarded to the local output port instead.

### *5.2.1 Rerouting Principle*

The general idea of the *Software-based Packet Rerouting* method is to redirect packets to some *intermediate destination* node to bypass a shut down component. From the intermediate destination the packet then is sent to its actual destination using the routing on the network layer again. The intermediate destination is calculated by the *software routing* algorithm on the software layer. The information employed by the algorithm to calculate an intermediate destination and the required packet format for software-based rerouting are described in the following subsections.

### 5.2.1.1 Required Rerouting Information

For the calculation of alternative paths the software routing algorithm uses information available on different network layers, as shown in Figure 5.5. From

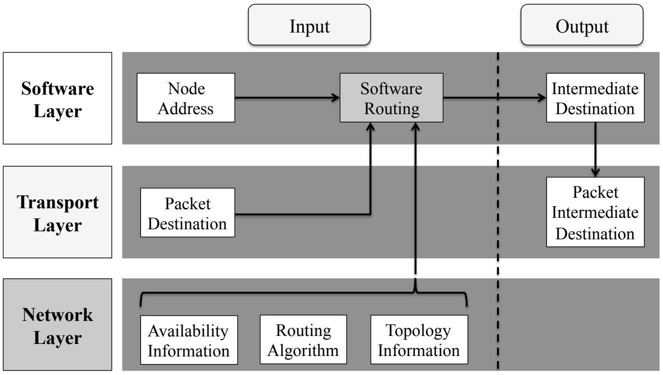


Fig. 5.5: Information employed by software routing algorithm.

the network layer, software routing requires the information about the routing algorithm and the NoC's topology. The granularity of the topology information depends on whether it is a regular or irregular topology. In case of an irregular topology, detailed information about the interconnection of network nodes is required, e.g. by means of an adjacency list. For regular topologies, the size (ring) of the topology or its dimension (mesh, torus) is sufficient. Furthermore, information about the availability of crossbar connections and links of a switch (*Availability Information*) is required to determine which output ports can be used for rerouting. This information is provided by the *Availability Status Communication Unit* (ACU) (cf. Subsection 5.1.1). While routing algorithm and topology is information that can be provided to the software routing during design time, the availability information has to be communicated to the software layer online as availability of resources may change during NoC operation.

To calculate a specific alternative path for a packet, software routing employs address-related information. This information is: the address of the node performing software routing (*node address*) and the destination address of the packet (*packet destination*), which define the start and end point of the alternative path. While the destination address is stored in the packet header available on the transport layer, the node address is information available on the software layer.

Using the above information software routing calculates an intermediate destination for a packet. On the transport layer the packet is updated with this information.

#### 5.2.1.2 Packet Format

To reroute a packet *pkt* to an intermediate destination and from there to the packet's original destination it is required that the addresses of both network nodes is stored in the packet header. For this purpose, the header must contain two separate fields: the *destination field* (pkt.dest) and the *backup field* (pkt.backup). The backup field contains the address of the original destination node and may only be set by the original sender of a packet. The destination field contains the address of the node to which the packet currently has to be forwarded to. The information stored in this field is considered by the routing implemented on the network layer. In case of a packet rerouting the destination field is overwritten by the software routing. Initially, when a packet is injected by its sender, both fields contain the destination address. By means of the two address fields a node *n* that receives the packet can determine its role for the packet, as shown in Listing 3.

**Listing 3** Role of Network Node *n*.

```
 1: if pkt.dest == pkt.backup then
 2:     if n.addr == pkt.dest then
 3:         node n is receiver of packet pkt
 4:     else
 5:         node n has to perform software routing
 6:     end if
 7: else
 8:     if n.addr == pkt.dest then
 9:         node n is intermediate destination
10:     else
11:         node n has to perform software routing
12:     end if
13: end if
```

If both addresses stored in the packet are identical, the packet is currently forwarded towards it destination (line 1). If address *pkt.dest* matches with the one of node *n* (*n.addr*), then *n* is the receiver of the packet (line 3). If *pkt.dest* and *n.addr* are different, the packet has to be routed in software as it could not proceed on its normal path due to an unavailable communication component (line 5). If both address fields are not identical this implies that the packet is currently redirected heading towards its intermediate destination. If *pkt.dest* and *n.addr* are identical, node *n* is the intermediate destination of packet *pkt* (line 9). If both addresses are not identical then the alternative path is unavailable as well, and thus, node *n* has to redirect the packet again (line 11).

In general, software routing allows packets to be sent back via their input port in order to bypass shut down links. However, sending back a packet must be avoided in case of a *dead end* situation as this would imply that the packet is repeatedly sent into the dead end. For this purpose the packet header features an additional bit (pkt.dead_end) to signal that a packet may not be sent back via the same port. To identify the input port of a packet in software, the address of the sender (pkt.send) is used. The dead end handling is explained in Subsection 5.2.2.1.

## 5.2.2 Software Rerouting

### 5.2.2.1 Rerouting Logic Overview

Every time a switch receives a packet *pkt* that cannot be forwarded by the routing implemented on the network layer, it is passed to the software rerouting method of the PE. The pseudo code of the packet rerouting algorithm is shown in Listing 4.

A packet *pkt* that has to be rerouted in software has to be stored in the PE's packet buffer until it is re-injected into the NoC again. For this reason, a packet can only be accepted for rerouting if the buffer is not full and can store the packet (line 2). If this is not the case, the packet has to be discarded and has to be retransmitted by the sender at a later time. If the packet can be accepted for rerouting, first, the set of all available ports *Avail_Ports* of the current node's switch is determined not including the local port (line 11). For this purpose, the *Availability Information* received from the *Availability Status Communication Unit* is employed (cf. Subsection 5.1.1). Subsequently, the input port *pkt_i* via which packet *pkt* was received at the current switch and the output port *pkt_o* to which packet *pkt* has to be forwarded are calculated according to the

**Listing 4** Software Rerouting Algorithm.

get_avail_ports(): returns all ports leading to an available link.
avail(): returns the availability status of a crossbar connection.
get_input(): determines the input port *pkt_i* where the packet was received.
get_output(): determines the output port *pkt_o* of the packet.
software_routing(): determines the intermediate destination.

```
procedure SOFTWARE_REROUTING_LOGIC($pkt$)
  if packet buffer full then                              ▷ check if packet can be accepted
    drop packet $pkt$
    return
  end if

  if $curr$ = intermediate destination then
    $pkt.dest := pkt.backup$;                             ▷ restore destination address
  end if

  $Avail\_Ports$ := get_avail_ports() $\setminus L$;      ▷ obtain all available ports
  $pkt\_i$ := get_input($pkt.send$);                      ▷ determine input port $pkt\_i$ and ...
  $pkt\_o$ := get_output($pkt.dest$);                     ▷ output port $pkt\_o$ of packet
  $pkt.send$ := $curr$;                                   ▷ set current node $curr$ as new sender of packet

  if $pkt.dead\_end$ then
    $Avail\_Ports := Avail\_Ports \setminus pkt\_i$;      ▷ remove disallowed port
    $pkt.dead\_end$ := false;
  end if

  if $pkt\_o \in Avail\_Ports \wedge$ avail($L \rightarrow pkt\_o$) then
    bridging of shut down crossbar connection             ▷ no rerouting necessary
    success := true;
  else
    success := software_routing($pkt$);                   ▷ perform software routing for packet $pkt$
  end if

  if $\neg$ success $\wedge$ $Avail\_Ports$ != $\varnothing$ then   ▷ dead end handling
    $out\_port$ := select($Avail\_Ports$);
    $pkt.dest$ := neighbor of $curr$ in direction $out\_port$;
    $pkt.dead\_end$ := true;
    success := true;
  end if

  if success then
    re-inject packet $pkt$                                ▷ rerouting possible
  else
    discard packet $pkt$                                  ▷ rerouting not possible
  end if
end procedure
```

routing algorithm used on the network layer. This information is determined by means of the sender address *pkt.send* (for *pkt_i*) or destination address *pkt.dest* (for *pkt_o*) found in the packet (line 12 and 13). Further, the packet's sender address is replaced by the address of the current node *curr* (line 14). This is required to determine the input port of the packet at another node in case the packet has to be rerouted a second time. If the *dead_end* field of the packet is set, it must not be sent back to the port via which it was received, and thus, port *pkt_i* is removed from set *Avail_Ports* (line 17). The dead end situation is explained later.

If both the output port *pkt_o* and the crossbar connection from local port *L* to *pkt_o* are available, this means that packet *pkt* could not be forwarded because the crossbar connection from the input *pkt_i* to output port *pkt_o* is shut down. In this case, packet *pkt* is not rerouted, since the shut down crossbar connection is bridged when *pkt* is re-injected using $L \rightarrow$ *pkt_o* (line 21 - 23). Once the packet is re-injected, it can be forwarded to its destination using the original path.

If either connection $L \rightarrow$ *pkt_o* or port *pkt_o* is not available, software routing has to calculate an alternative path for *pkt* (line 25). For this purpose, a network node is determined to become the packet's intermediate destination. A node may become an intermediate destination if from this node the packet can reach its original destination using the routing on the network layer without passing the shut down crossbar connection or link. If an appropriate intermediate destination is found by the software routing, its address is stored in the destination field of the packet.

Software routing may fail to redirect a packet. This is the case if the packet has reached a *dead end*. A dead end is a situation where the packet neither can be sent via the intended link towards its destination nor via one of the redirect links because they are shut down as well. To avoid the packet being stuck, the rerouting logic implements dead end handling (line 28 - 33). A dead end situation is solved at node *curr* by sending the packet back to the neighboring switch, from which it was received earlier. At the neighbor it has to be ensured that the packet is not sent to the port that leads back to *curr*. For example, when *curr* sends the packet back to its western neighbor, the usage of the eastern port must be prohibited as this would return the packet to node *curr*. To signal a packet must not be sent back, the header field *dead end* is set.

The packet *pkt* can be re-injected into the NoC if it either can bridge the shut down crossbar connection or if an intermediate destination, i.e. an alternative path, was successfully determined. Software rerouting for packet *pkt* fails when:

1. none but the local port is available because all other ports are shut down or must not be used, i.e. *Avail_Ports* == ∅, and when
2. software routing cannot determine an alternative path.

In both cases packet *pkt* has to be discarded (line 35 - 39).

Redirecting packets to an intermediate destination and from there to the packet's actual destination violates the routing rules of the routing implemented on the network layer. Thus, this involves the risk of deadlocks. To avoid deadlocks, a redirected packet is removed from and later re-injected into the network at its intermediate destination. This splits the alternative path into two subpaths:

1. from the rerouting node to the intermediate destination and
2. from the intermediate destination to the actual destination.

On each of the subpaths the packet is sent by using the network layer routing, and thus, no routing rules are violated. However, as in the case of software rerouting, a packet can only be accepted by an intermediate destination if the packet can be stored in the PE's packet buffer. Otherwise the packet has to be discarded. When an intermediate destination can accept the packet, the original destination is restored using the address in the backup field of the packet: *pkt.dest* := *pkt.backup* (line 8). The packet then is re-injected into the network and is sent to its destination using the routing on the network layer.

#### 5.2.2.2 DOR XY Software Routing

The software routing method (*software_routing* line 25) may implement any routing algorithm. In this subsection DOR XY software routing is presented as specialization of software routing is presented, applicable for NoCs with mesh topology. The employed network layer information (cf. Figure 5.5) of this specialization is:

- *Routing Algorithm*: the XY routing rule.
- *Topology Information*: the x- and y-dimension of the mesh topology, and
- *Availability Information*: the availability status of crossbar connections and links of the local switch.

The pseudo code of DOR XY software routing is shown in Listing 5.

Generally, in mesh topology a packet can be blocked by an unavailable communication resource while being forwarded either in x- or y-direction. Thus, when a packet is blocked in one direction it has to be redirected in the

**Listing 5** Software Routing.

avail(): returns availability status of a crossbar connection or a link.
exist_*dir*_neighbor(): returns *true* if neighbor in direction $dir = N, E, S, W$ of a node exists.

```
 1: procedure SOFTWARE_ROUTING($pkt$)
 2:    $success$ := $true$;
 3:    $dest$ := $pkt.dest$
 4:    $curr$ := this;                                   ▷ determine destination and current node
 5:
 6:    if $curr.x == dest.x$ then
 7:       if exist_E_neighbor($dest$) $\wedge$ avail(L $\rightarrow$ E) $\wedge$ $E \in Avail\_Ports$ then
 8:          $pkt.dest$ := east neighbor of $dest$;
 9:       else if exist_W_neighbor($dest$) $\wedge$ avail(L $\rightarrow$ W) $\wedge$ $W \in Avail\_Ports$ then
10:          $pkt.dest$ := west neighbor of $dest$;
11:       else
12:          $success$ := false;                                        ▷ Software routing failed
13:       end if
14:    else
15:       if exist_N_neighbor($curr$) $\wedge$ avail(L $\rightarrow$ N) $\wedge$ $N \in Avail\_Ports$ then
16:          $pkt.dest$ := north neighbor of $curr$;
17:       else if exist_S_neighbor($curr$) $\wedge$ avail(L $\rightarrow$ S) $\wedge$ $S \in Avail\_Ports$ then
18:          $pkt.dest$ := south neighbor of $curr$;
19:       else
20:          $success$ := false;                                        ▷ Software routing failed
21:       end if
22:    end if
23:
24:    return $success$;
25: end procedure
```

corresponding other direction, first, to bypass the resource. DOR XY software routing uses the information about the forwarding direction of a packet to determine an intermediate node. This information is obtained by comparing the x-coordinate of the packet's destination address *dest* with the one of the current node's address *curr*. If both addresses have the same x-coordinate the packet was blocked due to an unavailable resource in y-direction, and thus, the packet is redirected in x-direction ($curr.x - 1$ or $curr.x + 1$) (line 6). If both x-coordinates differ, the packet is redirected in y-direction (line 14).

To meet the requirement that from an intermediate node the packet can reach its destination using the network layer routing (cf. Subsection 5.2.2.1), DOR XY software routing uses either the east or west neighbor node of *dest* as the intermediate node when redirecting the packet in x-direction (line 7 - 10). For redirecting the packet in y-direction the north or south neighbor node of *curr* is used (line 15 - 17). The selection of a node as intermediate destination is based on the employed topology and availability information. A node is selected if:

- it exists,
- the crossbar connection at node *curr* from local port to the output port of the switch is available required to reach the intermediate node, and
- if sending the packet to the corresponding output port is not permitted due to a dead end situation (cf. Subsection 5.2.2.1).

If no intermediate destination can be determined, the DOR XY software routing fails (line 12 and line 20).

### *5.2.3 Evaluation*

The software rerouting method (cf. Listing 4) using DOR XY software routing (cf. Listing 5) is evaluated for an 8x8 mesh. Simulations are carried out using different injection rates and a different number of shut down links. The setup of the NoC simulation model used is summarized in Table 5.2.

Table 5.2: Simulation Model Setup for Software Rerouting

| Parameter | Setup |
|---|---|
| Topology | 8x8 Mesh |
| Routing | XY |
| Arbitration | Round Robin |
| Switching | Wormhole |
| Packet Size | 5 Flits |
| Traffic Pattern | Uniform Random |

For the evaluation of the software rerouting method the *packet redirection rate* is determined. The packet redirection rate represents the ratio between the number of packets being successfully redirected to their destinations and the total number of packets requiring rerouting.

### 5.2.3.1 Packet Redirection Rate

Packets that are rerouted have to be consumed by at least two different network nodes, i.e. the node performing software rerouting and the intermediate destination. In both cases a packet may only be accepted if it can be stored in the packet buffer of the corresponding PE. If the packet cannot be stored in the buffer of one of the nodes then it has to be discarded and rerouting fails. In the following, the software rerouting method is evaluated:

- for cases with one, five, and ten randomly shut down NoC links,
- for different software buffer sizes, and
- for different flit injection rates.

For each case one hundred different link shutdown scenarios are simulated and for each scenario the packet redirection rate is determined. The average packet redirection rate for different injection rates using a software buffer that can store ten packets is shown in Figure 5.6.

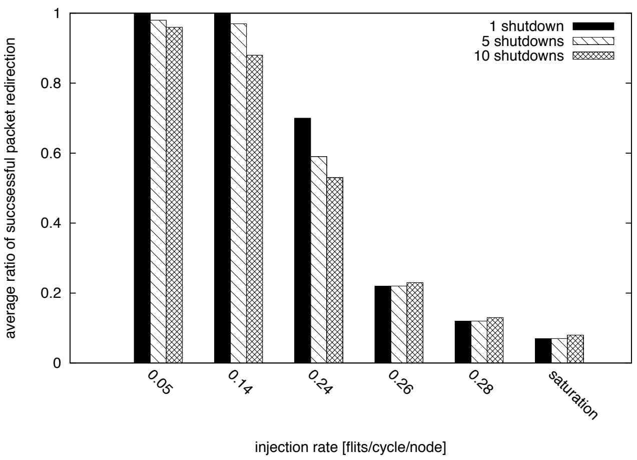


Fig. 5.6: Average packet redirection rate.

The results show that the packet redirection rate changes with the injection rate and the number of shut down links. For small (0.05 flits/cycle/node) and moderate (0.14 flits/cycle/node) injection rates software rerouting method is always able to successfully redirect packets to their destination in case of one shut down link. For five shutdowns approximately 97% of the packets requiring rerouting can be successfully redirected for both injection rates. While the redirection rate for ten shutdowns is comparable to the one of five shutdowns in case of the small injection rate, for the moderate injection rate it is reduced to 0.88.

For high injection rates ($\geq$ 0.24 flits/cycle/node) the number of successfully redirected packets strongly decreases for all shutdown scenarios. Due to the high load in the network, packets cannot be injected immediately but have to be stored in the packet buffer of the PEs for a longer time. As new packets arrive over time that have to be stored, packet buffers reach their limit, and thus, software rerouting and intermediate destination nodes can accept further packets less frequent. For injection rates near to (0.28 flits/cycle/node) or at saturation ($\sim$ 0.30 flits/cycle/node) the average packet redirection rate is only 0.12 or 0.07, respectively. This implies that around 90% of the packets are discarded while being redirected, and thus, those packets have to be retransmitted at a later time.

To evaluate the impact of the packet buffer size on the packet redirection rate, the evaluation of software rerouting method is repeated for software buffers that can store 25 and 100 packets, respectively. For this the same hundred shutdown situations have been simulated. The corresponding redirection rates are shown in Table 5.3. The results show that the increase of the packet

Table 5.3: Packet Redirection Rate

| Injection rate | buffer size 25 | | | buffer size 100 | | |
|---|---|---|---|---|---|---|
| [flits/cycle/node] | 1 | 5 | 10 | 1 | 5 | 10 |
| 0.05 | 1 | 0.98 | 0.96 | 1 | 0.98 | 0.96 |
| 0.14 | 1 | 0.97 | 0.88 | 1 | 0.97 | 0.89 |
| 0.24 | 0.7 | 0.59 | 0.53 | 0.71 | 0.6 | 0.54 |
| 0.26 | 0.22 | 0.22 | 0.23 | 0.22 | 0.23 | 0.23 |
| 0.28 | 0.12 | 0.12 | 0.13 | 0.12 | 0.13 | 0.13 |
| saturation | 0.07 | 0.07 | 0.08 | 0.07 | 0.08 | 0.08 |

buffer does hardly improve the packet redirection rate. For 25 buffer slots the measured rates are the same as in case of ten buffer slots. For very large packet buffers with 100 slots the measured maximum rate increase is only about 0.01. The reason for this very small increase is that the buffer slots are occupied by packets waiting for their injection, and thus, the buffer slots are not available for redirected packets.

In summary, the results show that software-based packet rerouting is only of limited applicability for NoCs with high network load because of the large number of packets that have to be discarded due to insufficient packet buffer size. For small and moderate injection rates, however, the software-based rerouting of packets is suitable to tolerate the shutdown of communication resources on the network layer.

#### 5.2.3.2 Implementation Costs

The code size of the software rerouting logic is only 5.7 KiB per network node when using DOR XY software routing. However, as shown in Subsection 5.2.3.1, packet redirection may fail because the packet has to be dropped. For this reason, an end-to-end flow control protocol is required that ensures retransmission of packets. e.g. the base protocol discussed in Subsection 4.2.1. In case of the base protocol, another 25.1 KiB are required per node.

## 5.3 Reconfigurable Hierarchical Routing

Software-based rerouting of packets as presented in Section 5.2 tolerates the shutdown of communication resources in the NoC without the need to change the network layer routing. However, for NoCs with a high network load software-based rerouting is no longer suitable because a large number of packets have to be discarded due to full packet buffers at the PEs (cf. Subsection 5.2.3.1). Instead of redirecting packets at PEs, the communication paths in the NoC have to be adapted to bypass shut down resources. This can be achieved by reconfiguration of the network layer routing.

During the reconfiguration process new communication paths are calculated for the altered NoC topology. In contrast to software-based rerouting that determines an alternative path for each packet individually, the newly calculated paths are used by all packets to bypass shutdowns. Furthermore, while software-based rerouting is always performed in situ when a packet cannot be sent via the designated output port of a switch, routing reconfiguration is only performed once when a component is shut down. A continuous execution, as in case of software-based rerouting, is not required.

In general, routing reconfiguration in a NoC is a critical and time consuming task. In addition to the calculation of communication paths, this task comprises the exchange of network status information as well as the update of the routing. During the reconfiguration process the NoC switches are either not available or are only available to a limited extent for data communication. Thus, the NoC's performance is degraded. The reconfiguration of routing in the whole NoC, however, is inappropriate if only a single link or switch is shut down. With respect to network performance, it is advantageous to reconfigure the routing only for switches in the surrounding of the shut down component while the routing of all other switches is kept unchanged. A further challenge for routing reconfiguration is its increasing complexity with increasing NoC size with respect to the required time and communication overhead of status information. A possibility to cope with these issues is to organize NoCs hierarchically and to replace the global reconfiguration mechanism with local ones applied to the smaller hierarchical units.

In the following, a reconfigurable routing for large scale NoCs organized into logical hierarchical units is presented that makes use of routing tables. The hierarchical units allow the routing to be reconfigured locally within each unit individually in case of a link or switch shutdown while the connectivity and deadlock-freedom is guaranteed in the entire network. The individual tasks of the associated reconfiguration process are distributed over multiple layers of the network. While on the network layer it is ensured that the status of network resources is provided to all network nodes within a hierarchical unit, the calculation process takes places in software. The update of the routing tables is achieved by interaction of software and network layer mechanisms.

The hierarchical organization of a NoC's topology is described in Subsection 5.3.1. The principles of the hierarchical routing as well as the data structure and algorithm used to calculate the routing are presented in Subsection 5.3.2. In Subsection 5.3.3 the adaptation of the routing and reconfiguration of the routing tables in case of a shutdown of a faulty component is presented. Evaluation results are provided and discussed in Subsection 5.3.4.

### *5.3.1 Hierarchical Organization of NoC Topologies*

The reconfigurable hierarchical routing makes use of a logical hierarchy. To construct the hierarchy for a given NoC topology $T$ with network nodes $V$ and links $E$, one or more neighbored network nodes are aggregated as hierarchical units $\{U_{1,0},...,U_{1,i}\}$ resulting in a partitioning of $V(T)$. This partitioning is represented by graph $T' = (V',E')$ where $V' = \{U_{1,0},...,U_{1,i}\}$. Two vertices

$U_{1,j}$ and $U_{1,k}$ are connected by an edge in $T'$ if in $T$ at least one link exists connecting one node $v_m$ of unit $U_{1,j}$ with node $v_n$ of unit $U_{1,k}$:

$$E'(T') = \Big\{ \{U_{1,j}, U_{1,k}\} | \exists v_m \in U_{1,j}, \exists v_n \in U_{1,k} : \{v_m, v_n\} \in E(T) \Big\} \tag{5.1}$$

The vertices of $T'$ can be further aggregated, and thus, defining hierarchical units of the next hierarchy level. This aggregation step can be repeated and each step defines the units $U_{h,i}$ of the next higher hierarchy level $h$. The exemplary graphs for three aggregation steps for topology $T$ (cf. Figure 5.8) are shown in Figure 5.7.

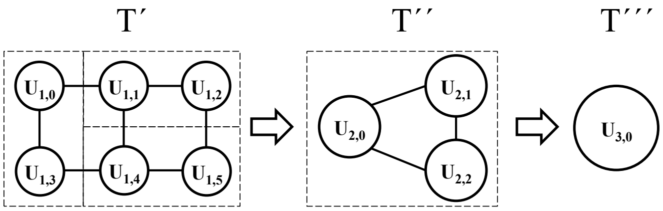


Fig. 5.7: Graph representations of aggregations.

On the highest level $h_{max}$, one single unit $U_{h_{max},0}$ exists that aggregates all units of the lower levels. Thus, $h_{max} = 1$ corresponds to the flat hierarchy network and the units on level $h = 0$ are the network nodes. The resulting hierarchically organized NoC with $h_{max} = 3$ for the above graphs is shown in Figure 5.8.

A unit $U_{h-1,j}$ on level $h-1$ is called a *subunit* of $U_{h,i}$ if $U_{h-1,j} \in U_{h,i}$. Unit $U_{h,i}$ is then called the *superunit* of $U_{h-1,j}$. In accordance with the partitioning, a hierarchical unit on level $h < h_{max}$ has exactly one superunit on level $h+1$.

## 5.3.2 Fault Tolerant Hierarchical Routing

Faults and the associated shutdown of communication resources occur during NoC operation and lead to a change of the NoC topology. In order to maintain communication, a reconfigurable routing has to exhibit the property that it can be adapted for arbitrary topologies. However, performing routing reconfiguration globally is a time consuming task in large scale NoCs (cf. Section 5.3).

Thus, to minimize the required time, reconfiguration has to be limited to the switches in the local surrounding of a shut down resource. To enable local reconfiguration in principle, it has to be ensured that the impact of routing reconfiguration on the connectivity and deadlock-freedom is contained within the surrounding of the shut down resource.

In order that a new routing takes effect in the NoC, the routing in the switches has to be updated. This requires a routing representation that allows the routing of a switch to be changed in principle. Additionally, an appropriate reconfiguration mechanism is required that replaces the old routing with the new one.

The hierarchically organized topology with units presented in Subsection 5.3.1 constitutes the base for the implementation of the proposed reconfigurable routing. To exploit the hierarchical NoC topology, the routing is hierarchically structured as well. Every hierarchical unit on level $h \geq 1$ defines an *internal routing* used for communication between its subunits. The overall hierarchical routing is composed of these internal routings.

To be able to calculate a routing for arbitrary topologies, *Up/Down* routing (cf. Subsection 2.2.5) is employed for all the internal routings. In case of a shutdown of a faulty component, only the internal routing of the affected hierarchical unit is reconfigured.

In each hierarchical unit on level $h \geq 1$ one of the network nodes is defined as the unit *manager*. The task of a manager is to collect information about shut down links and switches within its unit. By making use of this information, a manager is responsible to recalculate the internal routing and to incorporate

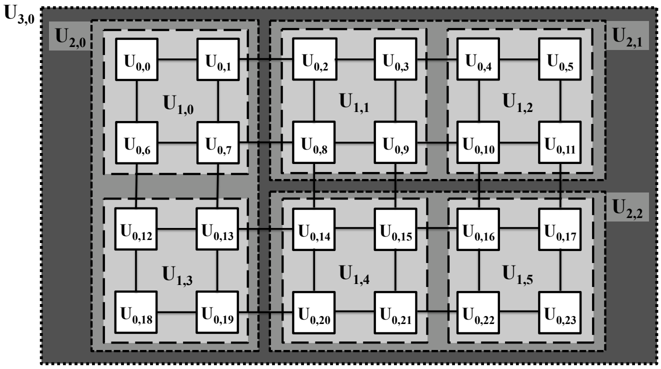


Fig. 5.8: Hierarchical mesh topology with $h_{max} = 3$.

the result into a hierarchical graph representation of the topology referred to as *Enhanced Topology Graph* (ETG) in the following. An ETG is used by the nodes within a unit to calculate the hierarchical routing. The calculated routing of a node is stored in a routing table of its switch.

Deadlock-freedom of the hierarchical routing is ensured by using virtual channels.

The routing principles as well as the structure of routing tables are presented in Subsection 5.3.2.1. The necessity of using virtual channels to ensure deadlock-freedom for the proposed hierarchical routing is discussed in Subsection 5.3.2.2. The generation of initial ETGs and the algorithm used to calculate the table entries is explained in Subsection 5.3.2.3 and Subsection 5.3.2.4.

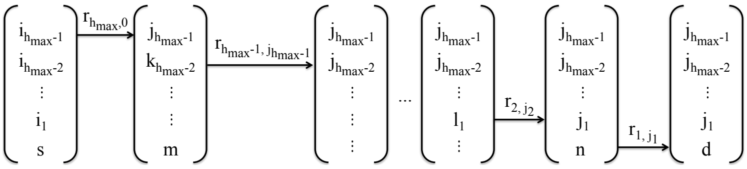
#### 5.3.2.1 Hierarchical Routing Principles and Representation

Each hierarchical unit $U_{h,i}$ on level $h$ is identified by its unique ID $i$. The address of network nodes is composed of the IDs of their hierarchical units in descending order, i.e. $(i_{h_{max}-1},...,i_0)$. For instance, the address of node $U_{0,23}$ is $(2,5,23)$ (cf. Figure 5.8).

The hierarchical routing works in a top-down manner selecting internal routings $r$ by descending hierarchy level. The hierarchical routing principle is depicted in Figure 5.9. Beginning with the internal routing $r_{h_{max},0}$ of unit $U_{h_{max},0}$, a packet from sender $s$ to destination $d$ is forwarded passing zero or more network nodes until it reaches a node $m$ with the same address ID $j_{h_{max}-1}$ as destination node $d$. This implies that $m$ and $d$ are encapsulated by the same hierarchical unit $U_{h_{max}-1,j_{h_{max}-1}}$ on level $h_{max}-1$. From now on, the packet is forwarded via zero or more nodes using the internal routing of unit $U_{h_{max}-1,j_{h_{max}-1}}$. When reaching a node whose IDs of level $h_{max}-1$ and $h_{max}-2$ match the ones of node $d$ the internal routing of the next lower level is used. This is repeated until node $n$ of the same unit $U_{1,j_1}$ as $d$ is reached. From $n$ internal routing $r_{1,j_1}$ is used to deliver the packet to destination $d$.

For instance, if in Figure 5.8 $U_{0,0}$ communicates with $U_{0,23}$, the packet is forwarded using the internal routing $r_{3,0}$ until it enters $U_{2,2}$. The packet then is further forwarded using $r_{2,2}$. When it reaches $U_{1,5}$, the packet is delivered to $U_{0,23}$ by employing the internal routing $r_{1,5}$.

The hierarchical routing is represented by routing tables implemented in each switch on the network layer. In contrast to flat network tables that contain one entry per network node, the hierarchical routing table $\mathcal{T}_m$ of node $U_{0,m}$ only contains entries for those network nodes that share the same level 1



$i_{h_{max}-1}$ $i_{h_{max}-2}$ ⋮ $i_1$ s
$r_{h_{max},0}$
$j_{h_{max}-1}$ $k_{h_{max}-2}$ ⋮ ⋮ m
$r_{h_{max}-1,\, j_{h_{max}-1}}$
$j_{h_{max}-1}$ $j_{h_{max}-2}$ ⋮ ⋮ ⋮
...
$j_{h_{max}-1}$ $j_{h_{max}-2}$ ⋮ $l_1$ ⋮
$r_{2,\, j_2}$
$j_{h_{max}-1}$ $j_{h_{max}-2}$ ⋮ $j_1$ n
$r_{1,\, j_1}$
$j_{h_{max}-1}$ $j_{h_{max}-2}$ ⋮ $j_1$ d

Fig. 5.9: Hierarchical routing principle.

superunit including an entry for $U_{0,m}$. All other entries refer to hierarchical units. Let $U_{h,[m]}$ denote the hierarchical unit on level $h$ that encapsulates node $U_{0,m}$. Thus, routing table $\mathscr{T}_m$ contains entries for all hierarchical units in $U_{h,[m]}$ except unit $U_{h-1,[m]}$.

$$\mathscr{T}_m = \{U_{0,m}\} \cup \bigcup_{h=1}^{h_{max}} U_{h,[m]} \setminus \{U_{h-1,[m]}\} \tag{5.2}$$

Table 5.4 shows the exemplary routing table for node $U_{0,23}$ for the hierarchical organized topology depicted in Figure 5.8. For each destination the table contains the output port and the VC that have to be used when forwarding the packet. The necessity of the VC information is explained later in Subsection 5.3.2.2. A packet for a destination node not being listed in the table is

Table 5.4: Routing Table of Node $U_{0,23}$

| Destination | Output port | VC |
|---|---|---|
| $U_{2,1}$ | north | $vc_{2,up}$ |
| $U_{2,0}$ | west | $vc_{2,down}$ |
| $U_{1,4}$ | west | $vc_{1,up}$ |
| $U_{0,16}$ | west | $vc_{1,down}$ |
| $U_{0,17}$ | north | $vc_{1,down}$ |
| $U_{0,22}$ | west | $vc_{1,down}$ |
| $U_{0,23}$ | local | $vc_{1,down}$ |

forwarded to the unit that contains the node. The unit is identified by means of the node's hierarchical address.

In a flat hierarchy network all routing tables have the identical content offering the possibility to use the ID of a node as index to access the corresponding table entry. In case of a hierarchical organized topology, however, only nodes within the same level 1 unit have identical tables. Thus, in routing tables of nodes situated in different units, different entries may be located at the same index. For this reason, it is assumed that tables are implemented as content-addressable memory.

#### 5.3.2.2 Deadlock-Freedom

As mentioned in Section 5.3.2, the hierarchical routing is composed of the internal routings of the hierarchical units. For the communication between two network nodes being situated in different units at least two internal routings of different levels are used (cf. Subsection 5.3.2.1)

All internal routings are calculated independently from each other. While each internal routing on level $h$ is deadlock-free, this is no longer the case when internal routings of more than one level are used at the same time. The reason for this is that packets using internal routings of different levels share the same link channels during NoC operation. Thus, deadlock-freedom is no longer guaranteed.

Deadlock-freedom of the hierarchical routing is achieved by the use of virtual channels (VC). For each hierarchy level $h \geq 1$ the hierarchical routing makes use of VCs for *Up* and *Down* direction, i.e. $vc_{up}$ and $vc_{down}$. Each VC has a rank. The rank of a VC depends of its level: the higher the level of a VC the higher its rank. Within one level, the *Up* VC has a higher rank than the *Down* VC. VCs have to be allocated by descending rank only. The dependency graph for the used VC allocation scheme is shown in Figure 5.10.

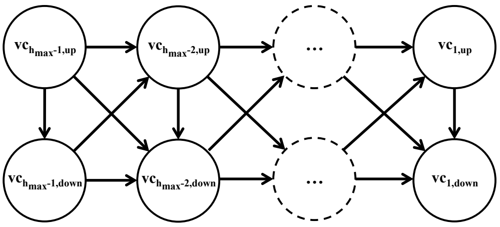


Fig. 5.10: Dependency graph of VC allocation scheme.

As long as an internal routing of level $h$ is used, the corresponding VCs $vc_{h,up}$ and $vc_{h,down}$ of that level are used. When changing to an internal routing of the next lower level, VCs of that level have to be used as well. For a communication between two network nodes of the same level 1 unit always the lowest rank VC $vc_{1,down}$ is used. The proof that the utilization of VCs together with the presented allocation scheme provides deadlock-freedom for any number of hierarchical levels can be found in the Appendix A.1. Simulations of the NoC model using the hierarchical routing have confirmed that the routing is deadlock-free.

#### 5.3.2.3 Enhanced Topology Graph

For calculating a hierarchical routing the hierarchy must be reflected by the data structure of the *Enhanced Topology Graph* (ETG) used by the calculation algorithm. An ETG is composed of vertices $v_{h,i}$ representing units of different hierarchy levels and is created by merging the topologies of hierarchical units.

Let graph $G_{h,u}$ be the representation of the abstract topology of unit $U_{h,u}$ whose vertices $\{v_{h-1,0},...,v_{h-1,i}\}$ represent the subunits of $U_{h,u}$. Analogous to the partitioning graphs in Subsection 5.3.1, two vertices in graph $G_{h,u}$ are connected by an edge if at least one link in topology T exists that connects nodes of both corresponding subunits (cf. Equation 5.1). Further, let $G_{l,[u]}$ denote the graph that aggregates $G_{h,u}$ on level $l$ and $v_{l-1,[u]} \in V(G_{l,[u]})$ be the representing vertex of the aggregation, where $h < l \leq h_{max}$. The vertex set of $ETG_{h,u}$ of unit $U_{h,u}$ then is defined by:

$$V(ETG_{h,u}) = \begin{cases} V(G_{h,u}) \text{ for } h = h_{max} \\ V(G_{h,u}) \cup \bigcup_{l=h+1}^{h_{max}} V(G_{l,[u]}) \setminus \{v_{l-1,[u]}\} \text{ for } 0 < h < h_{max} \end{cases} \tag{5.3}$$

Analogous to Equation 5.1, vertices of an ETG are connected by an edge if nodes within the corresponding units are connected by a link. Exemplary ETGs are shown in Figure 5.11.

The creation of initial ETGs follows a top-down approach and is performed by the unit managers. Starting with the creation of the ETG for the unit on highest level $h_{max}$, the resulting ETG is provided to the managers of the subunits of the next lower level. This is repeated until the managers of all level 1 units have created their ETG. The level-wise construction of $ETG_{1,5}$ for unit $U_{1,5}$ is shown in Figure 5.11. On highest level $h_{max}$, $ETG_{h_{max},0}$ is equivalent to graph $G$.

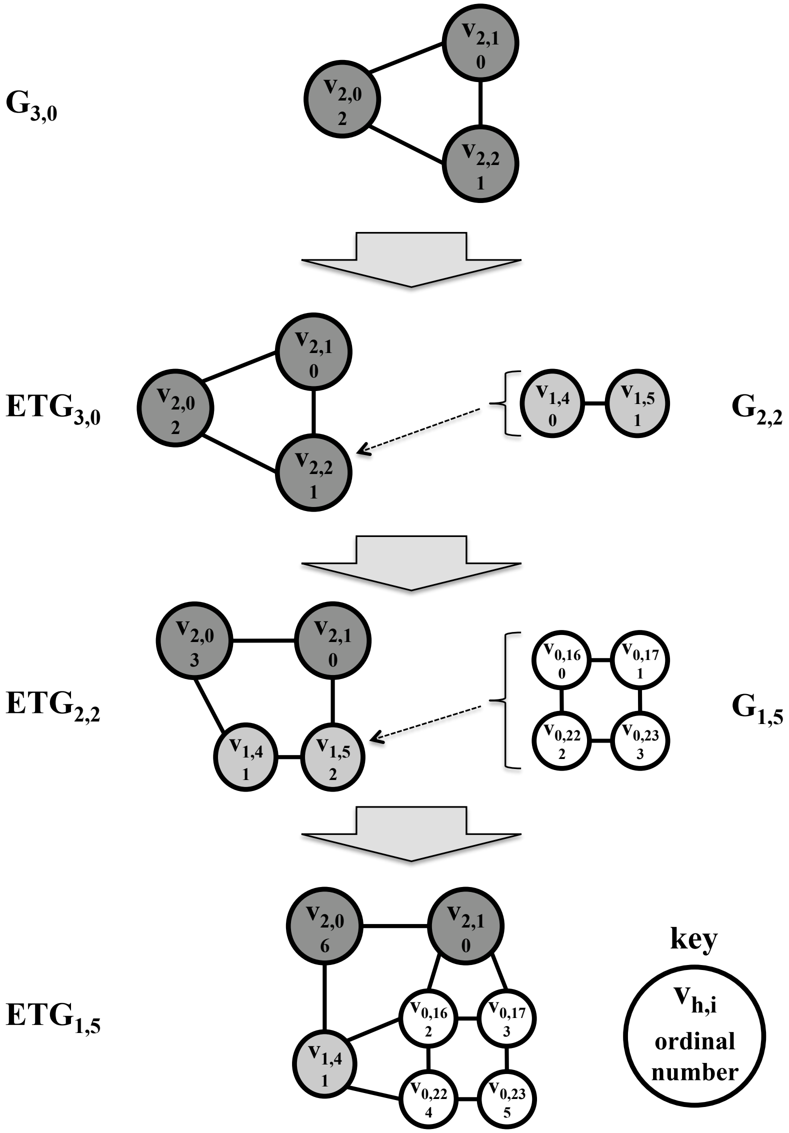


Fig. 5.11: Enhanced Topology Graph for $U_{1,5}$.

To create $ETG_{h,i}$ of a unit $U_{h,i}$, the manager extends $ETG_{h+1,j}$ received from the manager of the superunit. For this purpose, the manager replaces the vertex representing its own unit in $ETG_{h+1,j}$ by the unit's abstract topology graph $G_{h,i}$ and inserts the corresponding edges.

The initial graph $G$ is provided to the manager at design time. During NoC operation this graph is used to calculate the internal routing of a unit. For this purpose, the manager creates the breadth-first spanning tree of $G$ and assigns ordinal numbers to the vertices. The ordinal numbers are used during routing calculation to determine whether a corresponding unit can be reached via *Up* or *Down* direction. In accordance to the routing rules of *Up/Down* routing (cf. Subsection 2.2.5) unit $v_l$ is reached in *Up* direction from $v_k$ if it has a smaller ordinal number. Otherwise, $v_l$ is reached in *Down* direction.

When replacing a vertex $v_r$ in an ETG by those vertices of graph $G$, the order of vertices in the ETG has to be preserved. Otherwise, this would alter the internal routing of the superunit. For this reason, the ordinal numbers in the updated ETG have to be adapted. While all ordinal numbers being smaller than the one of $v_r$ are kept unchanged, all other ordinal numbers are increased by the number of vertices $|G|$ in graph $G$. Beginning with the ordinal number $o(v_r)$ of vertex $v_r$ the numbers from $o(v_r)$ to $o(v_r)+|G|-1$ are assigned to the vertices from graph $G$ according to their order. In Figure 5.11 the adaptation of ordinal numbers can e.g. be observed for the resulting $ETG_{2,2}$ after replacing $v_{2,2}$ of $ETG_{3,0}$ with the vertices of $G_{2,2}$.

To be able to insert the edges between the vertices of $ETG_{h+1,j}$ and the newly inserted vertices of graph $G_{h,i}$, the manager requires the adjacency information for all ports of each node within its unit. This information is obtained by comparing the hierarchical addresses (cf. Subsection 5.3.2.1) of two adjacent nodes beginning with ID $i_{h_{max}-1}$. The first ID of the addresses of both nodes which differs is the one relevant for the manager. For instance, the first ID in which $U_{0,16}$ and $U_{0,10}$ differ is $i_2$ (cf. Figure 5.8), and thus, the northern port of $U_{0,16}$ is connected to $U_{2,1}$. Just as the initial graph $G$, the adjacency information is provided to managers at design time.

#### 5.3.2.4 Routing Calculation

For the calculation of the hierarchical routing, the ETGs of level 1 are employed whereas all network nodes within unit $U_{1,i}$ use $ETG_{1,i}$.

To calculate the routing a modified Dijkstra algorithm is used that considers the *Up/Down* routing rules. The algorithm is shown in Listing 6. Bold line numbers identify lines containing an extension compared to the original Dijkstra algorithm (cf. [89] p. 198). Input parameters to the algorithm are the ETG and a starting vertex $v_{start}$. The starting vertex corresponds to the network node for which the routing is computed, e.g. for $ETG_{1,5}$ $v_{start}$ can be $v_{0,16}$, $v_{0,17}$, $v_{0,22}$, or $v_{0,23}$ (cf. Figure 5.11). As result the algorithm returns valid paths from $v_{start}$ to all other vertices of the ETG according to the *Up/Down* routing rules.

For the calculation, the algorithm makes use of two vectors:

- *cost*: If a vertex $v$ can be reached from the starting vertex $v_{start}$, the corresponding element $cost[v]$ contains a finite number. If $v$ cannot be reached $cost[v]$ is infinite.
- *pred*: If a vertex $v$ was reached by the modified Dijkstra algorithm, $pred[v]$ contains the ID of the adjacent vertex $v_a$ via which $v$ was reached; else no ID is stated. Starting at element $pred[v]$, the path between $v_{start}$ and $v$ can be determined by backtracking. In the following, $v_a$ is denoted as predecessor of $v$.

Furthermore, a queue (*sorted_Q*) is used in which vertices are sorted according to ascending costs.

After initialization of vectors and the queue (line 2 - 8) breadth-first search is performed on the ETG. In contrast to the original Dijkstra where a vertex can be reached from another one if both nodes are connected by an edge, in the presented modified Dijkstra algorithm for the hierarchical routing a vertex is only reachable if this does not lead to a violation of the *Down* to *Up* turn restriction.

For this reason, at the currently visited node $u$ the algorithm determines the direction via which $u$ was reached ($down_{pred}$; line 12) and the directions to the not yet visited neighbor vertices $w$ of $u$ ($down_{next}$; line 14). To determine the direction, the algorithm makes use of the ordinal number available for each vertex in the ETG. In general, a vertex is reached in *Down* direction if its predecessor has a smaller ordinal number; otherwise it is reached in *UP* direction.

By comparing directions $down_{pred}$ and $down_{next}$, the algorithm is able to identify the possible change in direction when visiting $w$. A neighbor vertex $w$ can only be reached from $u$ if either:

- $u$ and $w$ are representing units of the same level and the *Down* to *Up* turn restriction is not violated or
- if vertex $w$ represents a unit of a higher level (line 15).

**Listing 6** Dijkstra Algorithm for Hierarchical *Up/Down* Routing

```
 1: procedure DIJKSTRA_Up/Down( $ETG$, $v_{start}$ )
    o(): returns the ordinal number of a vertex
    lvl(): returns the level of a vertex
 2:   for all vertices $v$ in $ETG$ do                  ▷ initialization of vectors
 3:     $cost[v] := \infty$;                             ▷ cost to reach vertex
 4:     $pred[v] := \perp$;                              ▷ predecessor of vertex
 5:   end for
 6:   $cost[v_{start}] := 0$;          ▷ start vertex has no cost nor predecessor
 7:   $pred[v_{start}] := v_{start}$;
 8:   $sorted\_Q.insert(v_{start})$;        ▷ insert start vertex to sorted queue
 9:
10:   while not all vertices visited do
11:     $u := sorted\_Q.pop\_front()$                    ▷ get first vertex in queue
12:     $down_{pred} := (o(pred[u]) < o(u))$
13:     for all neighbors w of u do
14:       $down_{next} := (o(u) < o(w))$
15:       if $lvl(u) = lvl(w) \wedge (\neg(down_{pred} \wedge \neg down_{next})) \vee (lvl(u) < lvl(w))$ then
16:         $new\_cost := cost[u] + 1 + weight * lvl(w)$
17:         if $new\_cost < cost[w]$ then
18:           $cost[w] := new\_cost$;
19:           $pred[w] := u$;
20:           $sorted\_Q.insert(w)$
21:         end if
22:       end if
23:     end for
24:   end while
25:   return pred                                        ▷ return predecessor vector
26: end procedure
```

The *Down* to *Up* turn restriction applies only for vertices of the same level. An edge connecting two vertices of different levels is defined to belong to the higher of the two levels, and thus, in the latter case, turn violation check is omitted.

The cost to reach vertex $w$ is composed of the cost to reach $u$ plus one and the weighting factor *weight* multiplied by the level of $w$ (line 16). The higher the level of a vertex, the more network nodes are normally represented by this vertex. Thus, a path containing many vertices of higher levels is longer than one with vertices of lower levels. The weighting is done to constrain a path to consist of vertices of the lowest possible levels as this usually results in shorter paths. As the number of network nodes being represented by a vertex of level 1 or above is unknown to the algorithm, a favorable value for *weight* is the total number of available network nodes in the NoC as this corresponds to the maximum possible number of network nodes within a hierarchical unit.

In compliance with the original Dijkstra algorithm, $u$ becomes the predecessor of $w$ if the costs to reach $w$ via $u$ are smaller than the existing costs $cost(w)$. In this case, if not already stored, $w$ is added to the queue (line 17 - 21). The algorithm is repeated until all vertices of the ETG are visited.

After routing calculation, the VCs have to be determined used by the start node $v_{start}$ for the communication with the possible destinations $v_{dest}$. The pseudo code of the algorithm is shown in Listing 7. As input parameters the

**Listing 7** Algorithm to Determine VCs

```
 1: procedure DetermineVCs( $ETG$, $pred$, $v_{start}$ )
    o(): returns the ordinal number of a vertex
    lvl(): returns the level of a vertex
 2:   for all vertices $v$ in $ETG$ do
 3:     $v_{dest} := v$
 4:     $w := v$
 5:     if $lvl(w) = 0$ then
 6:       $vc\_vec[v_{dest}] := vc_{1,down}$
 7:     else
 8:       while $lvl(w) = lvl(v_{dest})$ do
 9:         if $o(w) < o(v_{start})$ then
10:           $vc\_vec[v_{dest}] := vc_{lvl(w),up}$
11:         else
12:           $vc\_vec[v_{dest}] := vc_{lvl(w),down}$
13:         end if
14:         $w := pred[w]$
15:       end while
16:     end if
17:   end for
18:   return vc_vec
19: end procedure
```

algorithm needs the ETG of level 1, the starting vertex $v_{start}$, and vector $pred$ of the modified Dijkstra algorithm. The VC used for communication is defined by the level of the destination. If level $h = 0$ then always VC $vc_{1,down}$ is used (line 6) as this is a communication between two nodes of the same level 1 unit, i.e. $v_{start} \in U_{1,i} \wedge v_{dest} \in U_{1,i}$.

For a communication to a unit of level $h > 0$, either $vc_{h,up}$ or $vc_{h,down}$ has to be chosen. The VC is defined by the first vertex on a path that has the same level as destination $v_{dest}$. To find this vertex, the algorithm searches a path in reverse direction starting with $v_{dest}$. By comparing the ordinal number of vertex $v$ with the one of the starting vertex $v_{start}$ the correct VC is determined (line 8 - 15). After completion of the algorithm, the VCs of all destinations are available (line 18).

The output port at $v_{start}$ used to reach $v_{dest}$ is defined by the first vertex on the path following $v_{start}$. The ports and VCs determined by the algorithm for $U_{0,23}$ using $ETG_{1,5}$ (cf. Figure 5.11) are shown in Table 5.4.

### *5.3.3 Cross-Layer Routing Reconfiguration*

So far, the generation of the initial ETG and the used algorithm used for routing calculation has been described in Subsection 5.3.2.1 and Subsection 5.3.2.4. In the following, the reconfiguration process of the hierarchical routing is described that is carried out when a faulty link or switch is shut down.

The adaptation of routing requires that the routing's reconfiguration process is triggered in order to maintain NoC operation. In general, the routing reconfiguration process can be split into three different phases:

1. *Information phase*: generation of required fault information for routing adaptation, e.g. location or type of a fault, and communication of the information to those NoC components that are involved in the routing adaptation process.
2. *Recalculation phase*: adaptation of routing based on the received fault information.
3. *Reconfiguration phase*: replacement old routing with new routing and activation of the new one.

The effort and complexity of the last phase mainly depends on the design and implementation of the routing. For instance, if the routing features precalculated alternative communication paths, the recalculation phase equals the activation of these paths. For other routings, the alternative paths are calculated in situ when a fault or shutdown is reported. Furthermore, the phases may not be strictly separable but may interleave.

In general, the required mechanisms for each of the phases can be implemented in hardware. Especially for the in situ routing calculation this, however, has the disadvantage that a hardware unit is required that implements the calculation algorithm. In addition to the required implementation overhead for the unit it is also prone to faults. The failure of this unit implies that the routing can no longer be calculated.

The reconfiguration mechanism for the hierarchical routing presented in this section implements the mechanisms of the three phases on different network layers, as shown in Figure 5.12. While the mechanisms for generation and communication of fault information as well as for reconfiguring the routing tables are implemented in hardware, the adaptation of the ETG and calculation of routing table entries are tasks performed in software. When a faulty

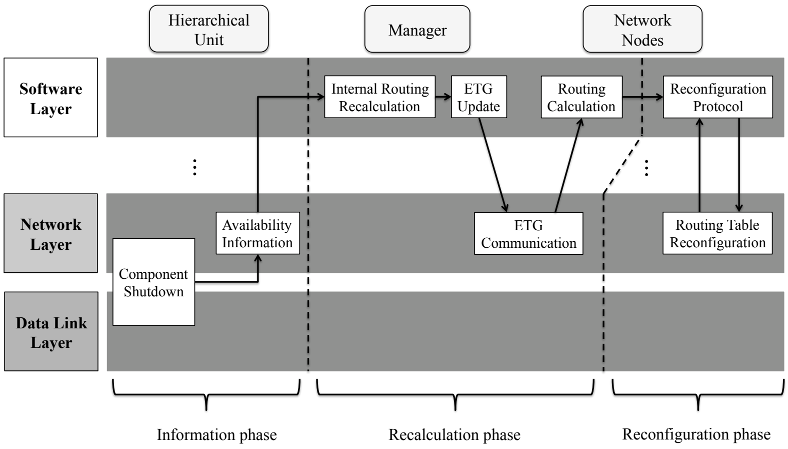


Fig. 5.12: Cross-layer reconfiguration process.

component is shut down, during the information phase all network nodes within the corresponding hierarchical unit are notified about it. This information is used by the manager in the recalculation phase to recalculate the unit's internal routing using graph $G$. Subsequently, the manager updates the ETG according to the routing changes. The updated ETG is provided to all nodes within the unit and each one calculates its routing table entries in software. In the last phase, the reconfiguration of routing tables on the network layer is triggered by the software by sending the new table entries from a node's PE to a dedicated reconfiguration unit of its switch.

In Subsection 5.3.3.1, the requirements of the reconfiguration process of the hierarchical routing are presented. The implementation of each phase is described in the subsections 5.3.3.2 to 5.3.3.4.

#### 5.3.3.1 Requirements of Reconfiguration Process

The reconfiguration process as presented in Figure 5.12 imposes requirements on the mechanisms used in the different phases to provide the interaction across layers. These requirements concern the switch design as well as the tasks performed in software.

***Information phase***: The shutdown of a communication resource alters the topology of a hierarchical unit. As this topology change is not reflected by the unit's current internal routing, packets being forwarded to the shut down resource can either not proceed, and thus, block the NoC or get lost. For this reason, all network nodes within the unit have to be informed that the current routing is no longer valid and that data communication temporarily is not reliable. In addition, the information about the shutdown has to be provided to the unit manager's software task to enable the calculation of a new internal routing. On the one hand, this requires the generation of the appropriate information about the shut down component. On the other hand, a communication mechanism is required that ensures that the information is provided to all network nodes within a unit even though the current routing is invalid.

***Recalculation phase***: When a manager receives the shutdown information it is required that the manager updates the unit's topology graph accordingly to be able to calculate the new internal routing for the unit. To enable the calculation of the new global routing, the manager has to integrate the topology change and the new internal routing to the ETG by removing vertices/edges and by updating ordinal numbers (cf. Subsection 5.3.2.3).

The overall time of the reconfiguration process shall be as small as possible to restore normal NoC operation quickly. The calculation of routing table entries by the manager for all network nodes within its unit, however, has a high complexity of $\mathcal{O}(|V|^3)$, and thus, results in a high calculation time. During this time all other nodes in the unit are idle as they have to wait for the update of their routing tables. Homogeneous NoCs offer the possibility to reduce the work load of a single core by distributing it to multiple cores. The proposed reconfiguration process makes use of the parallelism of a NoC in such a way that each network node calculates its own table entries. By this means, the complexity is reduced to $\mathcal{O}(|V|^2)$. For this purpose, it is required that the manager communicates the updated ETG to each node in its unit. For the communication of the ETG to the nodes it has to be considered that still the unit's internal routing is invalid. Thus, like for the communication of the shutdown information, a mechanism is required that ensures that the updated ETG is provided to all nodes within the unit.

***Reconfiguration phase***: As the routing table entries are calculated on the software layer, the entries have to be transported to the switch's routing table implemented on the network layer. Thus, the table entries have to be communicated by means of flits. On the network layer a reconfiguration mechanism is required that receives those flits, extracts the entry information, and updates the corresponding entry in the routing table. Transient and permanent faults, however, may affect the communication of table entries from software to network layer resulting in the corruption or loss of entries. This in turn results in an incomplete or invalid reconfigured table. For that reason, the reconfiguration mechanism has to feature logic to analyze flits and to request a retransmission of corrupted or missing table entries. A permanent fault in the connection between PE and switch, e.g. in the NI, prevents a successful table reconfiguration. The actual switch, however, may be fault free, and thus, its shutdown would be inappropriate. To ensure that the switch can still be used for communication, it is required that retransmissions of table entries can be requested from other network nodes.

***General requirement***: In general, switching from the old routing to the new routing is a critical task as this is prone to result in deadlocks in the network [88]. This is the case if both the old and the new routing are active in different switches at the same time. If the routing of a switch is only partially reconfigured this can lead to deadlocks as well. For this reason, the reconfiguration process has to consider this issue. This requires that a switch may only accept and send data if its routing table is completely reconfigured and data communication is enabled again by the reconfiguration process. To indicate whether a switch is completely reconfigured and may resume data communication a *reconfiguration status* is required that signals the end of reconfiguration.

#### 5.3.3.2 Information Phase

To generate the required availability information about the shut down component in the information phase, the *Availability Status Communication Unit* (ACU) presented in Subsection 5.1.1 is used. When a link becomes unavailable the ACU generates a single flit containing:

- the ID of the node incident to that link,
- the ID of the hierarchical unit to which the node belongs to, and
- the identification tag that identifies the shut down link.

As availability signals are always reset in both directions of a link (cf. Section 5.1.1), the ACU of both nodes incident to that link will generate a flit. On the one hand, this ensures that the shutdown of an entire switch is reported by the ACUs of the adjacent switches. On the other hand, both switches incident to the shut down link may be situated in two different hierarchical units. In this case, the generation of the availability information flit by both ACUs ensures that the manager nodes of both units are informed.

To avoid availability information flits being blocked by other data flits, they are communicated in a dedicated control VC. As the actual routing does no longer guarantee full connectivity between the nodes of the corresponding hierarchical unit, the router of a switch implements a flooding mechanism that is used for all flits received via the control VC. This guarantees that all nodes within a hierarchical unit receive the availability information.

Flooding, however, causes a large number of flit copies and thus leads to an increase of the network load. To minimize the increase of network load, each switch compares the unit ID found in the availability information flit with the one in its address (cf. Subsection 5.3.2.1). If both IDs do not match, the flit is dropped. Otherwise, a switch changes to the so-called *maintenance mode* and forwards the flit to all neighbors. By this means, flooding is limited to the unit affected by the shutdown and the boundary nodes of neighbored units.

While being in maintenance mode a switch neither accepts nor forwards any data flits. Already stored flits in the input buffers of a switch are dropped and all output port reservations are canceled. This ensures that all flits using the old invalid internal routing are removed from the NoC. If the flits are not dropped, this can lead to deadlocks as both, flits using the old routing and flits using the new one, may be present in the NoC at the same time when the routing is reconfigured. For dropping flits and canceling link reservations the mechanism of [32] is reused (cf. Subsection 5.1.2).

An availability information flit received at the manager's PE will trigger the recalculation phase.

#### 5.3.3.3 Recalculation Phase

The recalculation phase is composed of two parts. In the first part, the manager of a unit recalculates the internal routing and adapts the ETG while all other network nodes in the unit remain idle. In the second part, the ETG is communicated to each node within the unit and each node calculates its routing table entries on base of the received ETG. Internal routing recalculation, ETG adaptation, and calculation of routing table entries are tasks performed in software (cf. Figure 5.12).

The shutdown information generated during the information phase is used by the manager for recalculation of the internal routing if necessary and to update the ETG. The manager distinguishes between the following two cases:

1. the shutdown of links connecting two nodes that are situated in different units and
2. the shutdown of unit-internal links.

In the first case, the shutdown has no impact on the internal routing but on the routing of a unit of higher level, and thus, the manager skips recalculation of the internal routing of its unit and only removes the corresponding edge from the ETG. In the second case, however, the topology of the unit changes, and thus, recalculation of the internal routing is required.

For this purpose, the manager updates graph $G$, recalculates the *Up/Down* routing, and assigns the ordinal numbers to the vertices (cf. Subsection 5.3.2.3). The changes are then added to the ETG. Figure 5.13 shows an example for the recalculation of the internal routing of unit $U_{1,5}$ and the update of $ETG_{1,5}$ assuming the shutdown of the internal link between the nodes 16 and 17. Compared to the initial graph $G_{1,5}$ (cf. Figure 5.11) the edge between $v_{0,16}$ and $v_{0,17}$ is removed and the ordinal numbers are adapted.

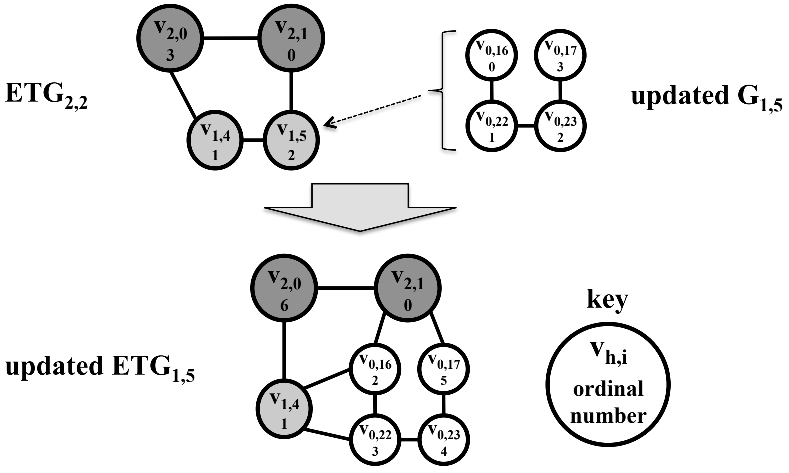


Fig. 5.13: Update of Enhanced Topology Graph $ETG_{1,5}$ after link shutdown.

The updated ETG is sent to the network nodes. Like availability information flits, the flits containing ETG information are forwarded using the flooding mechanism (cf. Subsection 5.3.3.2). When a network node has received all ETG flits it starts to calculate its new routing table entries using the modified Dijkstra algorithm presented in Subsection 5.3.2.4.

If at a network nodes the modified Dijkstra algorithm fails to find a path to a unit $U$ of higher level $h > 0$ in the ETG, the unit $U$ cannot be reached by the node. In that case, the internal routing of the superunit containing $U$ has to be adapted. For this reason, the node reports the connectivity problem to its manager which in turn informs the corresponding superunit manager of the next higher level. For this purpose, the manager creates the corresponding availability information and floods this information in the superunit of its unit. If required, the manager of the superunit again floods the information in the superunit of the next higher level. This is repeated until the manager of the superunit containing $U$ has received the information. When the ETG is updated, it is sent to the managers of the lower levels as in case of the initial ETG generation (cf. Subsection 5.3.2.3). Eventually, the updated ETG is provided to the network nodes for calculating their routing table entries.

When a node has calculated all table entries, it triggers the reconfiguration phase to update its routing table.

#### 5.3.3.4 Reconfiguration Phase

In the reconfiguration phase the table entries available at the PE of a node are written to the routing table of the node's switch. For each table entry, a separate *reconfiguration flit* is created on software side. These flits are sent consecutively to the switch. Each switch has an additional *Reconfiguration Unit* (RCU) to which the flits are forwarded. This unit extracts the information from the flits and updates the corresponding routing table entries.

The communication of table entries using flits is susceptible to transient and permanent faults. To avoid an invalid table reconfiguration due to flit corruption or loss, the communication between software and network layer is protected against faults by a cross-layer reconfiguration protocol. The protocol enables the RCU to request a retransmission of table entries from the software layer if the RCU cannot reconfigure the routing table successfully. The protocol has been designed and implemented in [3] and features two different reconfiguration modes:

1. *local reconfiguration* to control the reconfiguration between the RCU and the local PE, and
2. *remote reconfiguration* to allow the RCU to request reconfiguration from other network nodes.

Initially, the RCU is set to the local reconfiguration mode.

The protocol specifies that the start and the end of each reconfiguration flit stream from software to network layer are identified by additional start and end flits. To enable the RCU to detect flit corruption caused by transient or permanent faults, the protocol specifies that the reconfiguration flits have to be equipped with an error detection code on software side.

The network layer part of the protocol ensures the detection as well as the handling of faulty or missing table entries. Furthermore, the network layer protocol provides logic to toggle the reconfiguration modes. All of these mechanisms have been implemented in [3] as part of the RCU. To check reconfiguration flits for faults, the RCU is able to reuse existing checking units implemented at the input ports of the switch. If no checking unit is available, the RCU has to implement one itself.

To detect flit loss, the protocol specifies two mechanisms, i.e. timeout and the routing table reconfiguration status. For the timeout mechanism the RCU implements a hardware timer that is started once the start flit from software layer is received. If a timeout occurs before the end flit is received, at least one reconfiguration flit is considered to be lost.

The table reconfiguration status is represented by one additional bit per table entry. At the start of the reconfiguration phase all bits are reset. A status bit is set again when the corresponding flit was successfully received by the RCU and the entry is updated. If at the end of reconfiguration, i.e. the end flit was received or a timeout has occurred, one or more entries are still not marked as updated, the reconfiguration has failed and a retransmission of reconfiguration flits is requested.

If local reconfiguration fails three times the RCU switches to remote reconfiguration. To request remote reconfiguration the RCU sends one request to every neighbor of the node. The request, however, is only served by neighbor nodes within the same hierarchical unit as the requester. The reason is that only these nodes have the same ETG. From the set of remote reconfiguration offers from the neighbors the RCU selects one. If again the reconfiguration is not successful the RCU selects the next neighbor. Only in case the reconfiguration is not successful after requesting reconfiguration from all the neighbors, the RCU declares reconfiguration failure and shuts down the switch.

When the RCU has successfully finished the reconfiguration of the routing table, it sets the switch mode from maintenance to operative mode and the network node continues data flit communication.

### *5.3.4 Evaluation*

The evaluation of the proposed hierarchical routing is divided into the performance evaluation of the actual routing on the network layer (Subsection 5.3.4.1) and the performance evaluation of the cross-layer reconfiguration process (Subsection 5.3.4.2). For this purpose, different 16x16 mesh networks with flat hierarchy as well as $h_{max} = 3$ hierarchy levels are considered. In this work, all units of a level $h$ have the same size. For hierarchical units on level $h = 1$ unit sizes $s_1$ from $2x2$ to $8x8$ are considered. The size $s_2$ of level two units is always a multiple of the level one units and refers to the number of level one units that are encapsulated in x-dimension and y-dimension. The unit size $s_3$ always equals the network size (i.e. 16x16) and therefore is not mentioned in the following. The unit size of level one and two form the tuple ($s_1$,$s_2$), which is called the *Hierarchical Network Configuration* (HNC). The setup of the NoC simulation model used for performance evaluation is summarized in Table 5.5. Finally, in Subsection 5.3.4.3, the implementation overhead required to implement the hierarchical routing is investigated.

Table 5.5: Simulation Model Setup for Software Rerouting

| Parameter | Setup |
|---|---|
| Topology | 16x16 Mesh |
| Routing | Hierarchical Routing |
| Arbitration | Round Robin |
| Switching | Wormhole |
| Packet Size | 5 Flits |
| Traffic Pattern | Uniform Random |

#### 5.3.4.1 Routing Performance

To investigate the impact of unit sizes on the performance, the NoC model is simulated for 50 kcycles in saturation mode for different HNCs and the maximum throughput (TP $[\frac{received\,flits}{nodes \cdot cycles}]$) is measured. The measured TP results are shown in Figure 5.14.

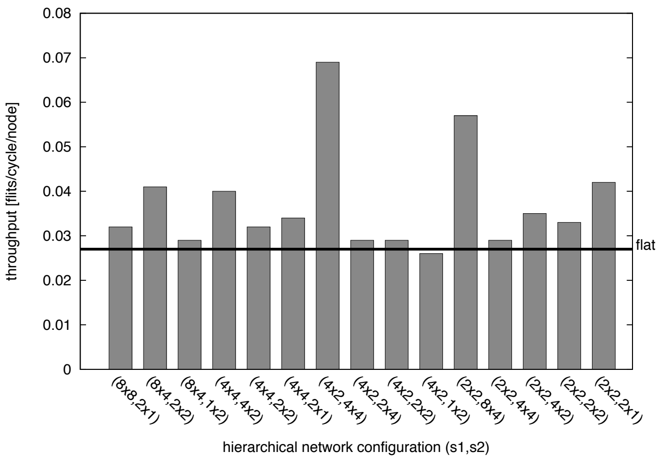


Fig. 5.14: Throughput of hierarchical routing.

The achievable throughput for the 16x16 flat hierarchy NoC is 0.027 flits per node per cycle. The results show, that for all HNCs the throughput increases except for (4x2,1x2). For (4x2,1x2), however, the throughput is comparable to the one of the flat hierarchy. The higher throughput of all other HNCs, especially for (4x2,4x4) and (2x2, 8x4), can be attributed to the more evenly distributed traffic of the hierarchical routing compared to *Up/Down* routing used in the flat hierarchy. This is exemplarily shown in Figure 5.15 for flat hierarchy (left) and HNC (4x2,4x4) (right). Figure 5.15 depicts the normalized values of communicated packets for each link for uniform random traffic pattern.

In *Up/Down* routing, many communication paths from sender to destination lead towards or via the root node [114]. Thus, the links near to the root node have a higher load than others. The high load at the links near to the root node causes a back pressure in the NoC which in turn causes long waiting times for packets before they can be forwarded. This behavior can be observed in Figure 5.14 for the flat hierarchy. Here the root node is situated in the upper left corner. The measured packet latency in case of the flat hierarchy is approximately 370 cycles on average.

As it can be observed in Figure 5.15 on the right, the hierarchical routing distributes the traffic more evenly in the NoC. On the one hand, this can be attributed to the partitioning of routing calculation into many independently unit-internal calculations. By using a different root node for calculation in each unit, communication is no longer aligned to only one root node as in case of the flat hierarchy. On the other hand, due to the additionally allowed *Up* to *Down* turns the hierarchical routing is less restrictive than *Up/Down* routing. Because of the better distributed traffic, the average packet latency is only 145 cycles. Compared to the flat hierarchy this corresponds to a latency reduction by factor 2.5.

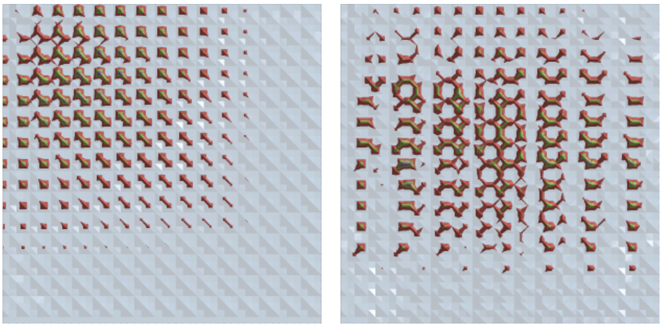


Fig. 5.15: Traffic distribution for flat and (4x2,4x4) HNC.

### 5.3.4.2 Performance of Cross-Layer Reconfiguration

For the performance evaluation of the reconfiguration mechanism of the hierarchical routing single link faults are injected into the NoC and the required reconfiguration time is measured for the flat network as well as for $h_{max} = 3$ levels. The measured times are compared to the distributed reconfiguration approach *Ariadne* [4] that reconfigures *Up/Down* routing for the flat hierarchy on the network layer. Evaluation has shown that the required reconfiguration time for the hierarchical routing is mainly composed of the time required for the following software tasks:

1. routing recalculation and $ETG$ update (cf. Subsection 5.3.3.3),
2. generation of routing table entries (cf. Subsection 5.3.2.4),
   a. computation of valid paths using modified Dijkstra algorithm and
   b. determining of VCs and output port.
3. level 2 reconfiguration.

Level 2 reconfiguration is omitted if only the routing within a unit on level 1 has to be adapted in order to restore connectivity. If the routing on level 2 has to be changed, then the corresponding time for recalculation of the internal routing and updating the $ETG$ adds to the total reconfiguration time. Compared to the software tasks, the time required for communicating the $ETG$ from the manager to all nodes within a unit is negligible. At maximum approximately 550 cycles are required for $ETG$ communication in the flat network.

To determine the time of a software task, the number of cycles is measured required by a 64 bit Linux computer system equipped with a 2.8 GHz Intel Core 2 Quad-CPU to perform the task. The total required reconfiguration time for the flat and hierarchical networks is shown in Figure 5.16. For better readability the required $ETG$ communication times for level 1 and level 2 reconfiguration are listed separately in Table 5.6.

Table 5.6: $ETG$ Communication Time [kcycles]

| HNC | flat | (8x8, 2x1) | (8x4, 2x2) | (8x4, 1x2) | (4x4, 4x2) | (4x4, 2x2) | (4x4, 2x1) | (4x2, 4x4) |
|---|---|---|---|---|---|---|---|---|
| Level 1 | 0.545 | 0.149 | 0.081 | 0.081 | 0.046 | 0.046 | 0.046 | 0.039 |
| Level 2 | - | 0.337 | 0.337 | 0.149 | 0.337 | 0.149 | 0.081 | 0.337 |
| $\Sigma$ | 0.545 | 0.486 | 0.418 | 0.230 | 0.383 | 0.195 | 0.127 | 0.376 |

| HNC | (4x2, 2x4) | (4x2, 2x2) | (4x2, 1x2) | (2x2, 8x4) | (2x2, 4x4) | (2x2, 4x2) | (2x2, 2x2) | (2x2, 2x1) |
|---|---|---|---|---|---|---|---|---|
| Level 1 | 0.039 | 0.039 | 0.039 | 0.021 | 0.021 | 0.021 | 0.021 | 0.021 |
| Level 2 | 0.149 | 0.081 | 0.046 | 0.337 | 0.149 | 0.081 | 0.046 | 0.039 |
| $\Sigma$ | 0.188 | 0.120 | 0.085 | 0.358 | 0.170 | 0.102 | 0.067 | 0.060 |

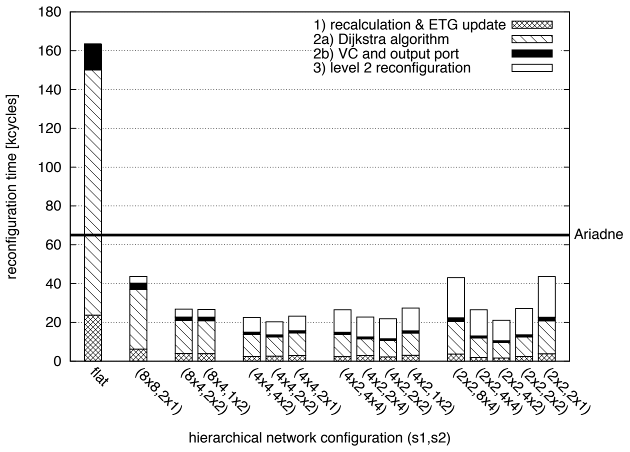


Fig. 5.16: Required time for routing reconfiguration.

For routing reconfiguration in the flat network, the cross-layer reconfiguration requires more than 160 kcycles. According to [4], *Ariadne* requires $|N|^2$ cycles for routing reconfiguration and thus approximately 65 kcycles are required for a 16x16 network (indicated by the horizontal line in Figure 5.16). This implies that for the flat network the time required by cross-layer reconfiguration is about 2.5 times higher compared to *Ariadne*. This is mainly caused by the time used for path computation on the software layer.

However, it can be observed in Figure 5.16 that in case of a hierarchical organized NoC the required time for reconfiguration is less than for *Ariadne*. This is even the case if the $ETG$ has to be updated on level 2. The reason for this reduction in time is the reduced number of vertices in the $ETG$ compared to the one of the flat hierarchy. The minimum reconfiguration time of approximately 20 kcycles is obtained for (4x4,8x8). For (2x2,8x4) less than 11 kcycles are required if the routing has only to be adapted on level 1.

### 5.3.4.3 Implementation Costs

The total implementation costs of RHR mainly consists of the software overhead for the routing recalculation and the hardware overhead for the Availability Communication Unit (ACU) and Reconfiguration Unit (RCU).

The software overhead differs between normal network nodes and manager nodes. While normal nodes only implement algorithms for routing calculation, i.e. the modified Dijkstra algorithm and the algorithm to determine the VCs (cf. Subsection 5.3.2.4), manager nodes additionally implement algorithms for internal routing recalculation and updating the ETG (cf. Figure 5.12).

For normal nodes, the total software overhead is approximately 31.2 KiB and the Dijkstra algorithm is about 65% of the total overhead. Compared to normal nodes, the software overhead for managers is about twice as high (60.1 KiB). The code sizes are shown in Table 5.7.

Table 5.7: Code Size of Recalculation Algorithms [KiB]

| Internal Routing Recalculation | ETG Update | Routing Calculation |
|---|---|---|
| 18.6 | 10.3 | 31.2 |

The ACU and the RCU, required for the cross-layer reconfiguration mechanism, are synthesized using the 45nm Nangate library [126] to investigate the hardware overhead compared to a 5-port baseline switch. The baseline switch features five VCs in total: four VCs for deadlock-free data communication and the control VC. For each VC the switch implements a VC buffer with three buffer slots per input port. Each buffer slot can store one flit of 36 bits width.

For the hierarchical routing the switch implements one routing table. Under the assumption that the node ID is used to access the routing table, a table entry for the flat hierarchy 16x16 mesh NoC only consists of 4 bits per entry representing the output port (3bits) and another bit for the RCU to determine whether a entry is reconfigured or not. For a hierarchically organized NoC with $h_{max} = 3$ hierarchy levels, each entry additionally requires 8 bits for the destination, two bits to represent the level, and two bits for the VC. Thus, in total one entry has 16 bits in this case. To obtain the area of routing tables for 45nm technology size, CACTI 5.3 from HP Labs [64] is used. The synthesis results are shown in Table 5.8.

The baseline switch without routing table has a size of approximately 0.0420 $mm^2$. The RAM size of the routing table varies between 0.0080 $mm^2$ (with hierarchy; 36 entries with 16 bit per entry) and 0.0148 $mm^2$ (flat hierarchy; 256 entries with 4 bits per entry) and depends on the number of entries. No sizes could be obtained for less than 36 entries as this comes below the minimum supported memory size of CACTI. In general, the number of table

Table 5.8: Synthesis Results

| Unit | Area [$mm^2$] |
|---|---|
| Baseline Switch | 0.0420 |
| Routing Table | 0.0080 / 0.0148 |
| Availability Communication Unit | 0.0014 |
| Reconfiguration Unit | 0.0015 / 0.0021 |

entries corresponds to the number of vertices in the ETG for a given HNC. The number of entries $\varepsilon$ for a HNC can be calculated by Equation 5.4.

$$\varepsilon = s_1 + \sum_{h=1}^{h_{max}-1} \frac{s_{h+1}}{s_h} - 1 \tag{5.4}$$

The number of table entries required for the different HNCs is shown in Table 5.9.

Table 5.9: Number of Routing Table Entries

| HNC | flat | (8x8, 2x1) | (8x4, 2x2) | (8x4, 1x2) | (4x4, 4x2) | (4x4, 2x2) | (4x4, 2x1) | (4x2, 4x4) |
|---|---|---|---|---|---|---|---|---|
| Entries $\varepsilon$ | 256 | 66 | 36 | 36 | 24 | 22 | 24 | 24 |

| HNC | (4x2, 2x4) | (4x2, 2x2) | (4x2, 1x2) | (2x2, 8x4) | (2x2, 4x4) | (2x2, 4x2) | (2x2, 2x2) | (2x2, 2x1) |
|---|---|---|---|---|---|---|---|---|
| Entries $\varepsilon$ | 18 | 18 | 24 | 36 | 22 | 18 | 22 | 36 |

The size of the ACU is independent of the number of hierarchy levels or table entries and has a fixed size of 0.0014 $mm^2$. The size of the RCU, however, depends on the number of routing table entries due to the increasing complexity of the logic used for updating table entries and for analyzing the reconfiguration status of the table. For 36 table entries the size of the RCU is about 0.0015 $mm^2$ and increases to 0.0021 $mm^2$ for 256 entries. For $h_{max} = 3$ hierarchy levels ACU and RCU account for about 6% of the total implementation costs.

## 5.4 Summary

The three different approaches presented in this chapter tolerate the impact of faults on the NoC communication on different network layers. Cross-layer fault tolerance is achieved by combining the management of communication resources methods of the data link layer with one of the routing-related methods of higher layers. For this purpose, availability status information of communication resources gathered on the data link layer is provided to the higher layer methods for rerouting packets or to reconfigure the routing.

Providing availability information to the software layer enables the rerouting and the reconfiguration task to be performed in software. Thus, no additional hardware component is required. Performing those tasks in software has the additional benefit that they can be migrated to another processing element in case of failure. When implemented in hardware, the migration is only possible if corresponding spare components exist. Spare components, however, contribute to the NoC's implementation costs.

The results of software-based rerouting and the reconfiguration of the hierarchical routing show that the software layer support to tolerate faults in the NoC has also drawbacks. In case of software-based rerouting, the successful redirection of packets can only be ensured in case of moderate injection rates. For higher injection rates, however, the rate of successful redirections strongly decreases due to full software packet buffers. This issue originates from the load information not being available on the software layer, and thus, it is not considered for the calculation of alternative paths. As the load changes dynamically, load information would have to be provided continuously to the software layer. This would result in a high communication overhead in the network.

The results for a flat hierarchy NoC show that cross-layer reconfiguration requires more time than routing reconfiguration performed in hardware. The reason for this is time overhead caused by the software routing recalculation process. By means of a hierarchical organization of a NoC, however, it is possible to decrease the amount of data that has to be processed during recalculation. This in turn leads to a reduction of the required recalculation time. Compared to the required time of the state of the art hardware reconfiguration process in [4] for flat hierarchy NoCs, the time for the cross-layer reconfiguration process of the proposed hierarchical routing is only about one-third.

# Chapter 6
# Combination of Cross-Layer Diagnosis and Fault Tolerance Methods

So far, the methods for cross-layer diagnosis and fault tolerance presented in Chapter 4 and Chapter 5 have been evaluated separately. In this chapter, both are evaluated in combination with regard to the impact on the system performance. For this purpose, each of the cross-layer (P+FD, P+SD, P+FD+SD) and standalone (P, FD, SD) diagnosis techniques is combined with *Software-based Packet Rerouting* (SBR) (cf. Subsection 5.2) as well as *Reconfigurable Hierarchical Routing* (RHR) (cf. Subsection 5.3). RHR is evaluated for $h_{max} = 3$ hierarchy levels using the hierarchical network configuration (4x2,4x4) (cf. Subsection 5.3.4.1). The resulting communication flow of the combinations is shown in Figure 6.1.

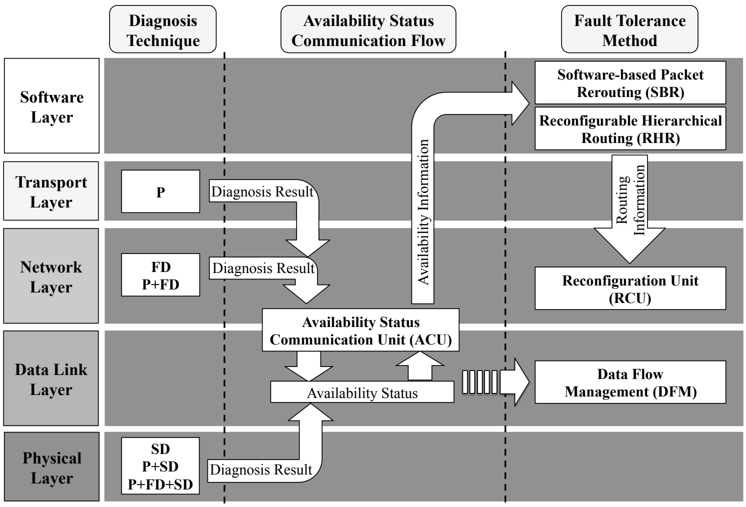


Fig. 6.1: Communication flow between diagnosis techniques and fault tolerance methods.

When diagnosis has located a faulty communication resource its availability status is reset to unavailable. For this purpose, P, FD, and P+FD send a flit with the diagnosis result to the *Availability Status Communication Unit* (ACU) of a switch to reset the corresponding availability status register. Because SD is performed on the physical layer, it is assumed that the register can be directly accessed, and thus, SD, P+SD, and P+FD+SD do not have to send a flit. In case of a change, the content of the availability status register is communicated to SBR or RHR using a flit. In combination with SBR the flit is sent to the local PE of a switch, whereas with RHR the flit is flooded to all PEs within a hierarchical unit. At the same time, the *Data Flow Management* (DFM) method ensures that the NoC is not blocked by flits when communication resources become unavailable. After routing recalculation is finished, the new routing information is sent to the *Reconfiguration Unit* (RCU) in case of RHR.

## 6.1 Evaluation

For the performance evaluation, a 16x16 mesh NoC is simulated for 500,000 cycles using uniform random traffic pattern and a network load near to saturation. The corresponding saturation injection rate is 0.138 flits/node/cycle for SBR and 0.077 flits/node/cycle for RHR, respectively. The base protocol's timeout $t_\delta$ is initialized to 40,000 cycles. The basic setup of the simulation model used for the evaluation is summarized in Table 6.1.

Table 6.1: Simulation Model Setup

| Parameter | Setup |
|---|---|
| Topology | 16x16 Mesh |
| Arbitration | Round Robin |
| Switching | Wormhole |
| Packet Size | 5 flits |
| Traffic Pattern | Uniform Random |
| Fault Tolerance Method | SBR; RHR |
| Injection Rate | SBR: 0.138; RHR: 0.077 |

The system performance is measured in terms of the achievable data throughput. The data throughput corresponds to the number of received data flits per node per cycle ([flits/node/cycle]). Acknowledgement flits are not taken into account. In the fault-free case, the achievable data throughput of

SBR and RHR equals the respective saturation injection rate. When the base protocol is activated, the data throughput is decreased to 0.115 flits/node/cycle (SBR) and 0.065 flits/node/cycle (RHR) because of the additional positive acknowledgements.

For the simulation of the NoC model with permanent faults, sets with one to five faults have been created. The time of occurrence of a fault as well as its location, i.e. either on a crossbar connection or on a link, are randomly chosen.

### *6.1.1 Cross-Layer Communication Overhead of Fault Tolerance Methods*

The cross-layer information exchange from network to software layer and vice versa causes additional communication overhead between the switch and the PE of a network node. In addition to the data packets, the cross-layer information occupies communication resources of the NoC such as the NI packet buffers. Thus, cross-layer information exchange reduces the number of data packets that can be sent or received by a PE.

In a first simulation, the cross-layer communication overhead in case of the occurrence of permanent faults is investigated for SBR and RHR. For this purpose, the number of flits is measured for each method that have to be exchanged between network and software layer in order to sustain communication in the NoC. The measured average overhead for SBR for one to five shut down links is shown in Table 6.2.

Table 6.2: Average Cross-Layer Overhead SBR [flits]

| | Shutdowns | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| Software Routing Node | 402,389 | 430,290 | 523,318 | 620,089 | 662,015 |
| Intermediate Destination Node | 200,039 | 215,438 | 247,374 | 239,751 | 252,718 |
| Total Overhead | 602,428 | 645,728 | 770,692 | 859,840 | 914,734 |

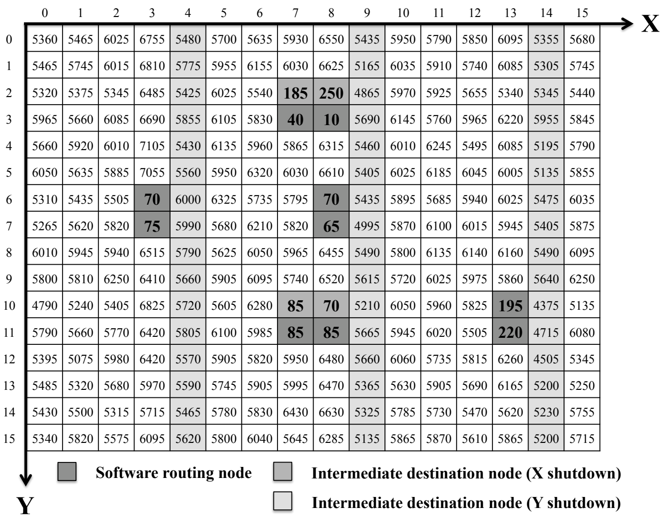
The cross-layer communication overhead for SBR is mainly composed of the flits of the:

- packets being routed in software and

- redirected packets that have to drained from and reinjected to the NoC to guarantee deadlock-freedom.

During simulation, for normal data communication (data + acknowledgements) about 21 million flits were exchanged between network and software layer. As shown in Table 6.2, for one shutdown additionally about 600,000 flits were exchanged between layers. In case of five shutdowns this number increases about factor 1.5 to more than 900,000 flits because more packets have to be redirected. This is an overhead of 2.8% and 4.3%, respectively, compared to the number of flits exchanged between layers for normal data communication.

The simulation has shown that the cross-layer communication overhead is not evenly distributed among all network nodes but mainly has to be handled by the nodes next to a shutdown component. To investigate the impact of the overhead on the data communication of these nodes, the number of data flits is measured that each node can inject to the NoC. Figure 6.2 shows the number of data flits for each node that are injected during an interval of 50,000 cycles in case of five shutdowns. A shutdown link is always situated between two nodes marked dark grey, i.e. software routing nodes. The results show that the


X
Y
Software routing node
Intermediate destination node (X shutdown)
Intermediate destination node (Y shutdown)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 5360 | 5465 | 6025 | 6755 | 5480 | 5700 | 5635 | 5930 | 6550 | 5435 | 5950 | 5790 | 5850 | 6095 | 5355 | 5680 |
| 1 | 5465 | 5745 | 6015 | 6810 | 5775 | 5955 | 6155 | 6030 | 6625 | 5165 | 6035 | 5910 | 5740 | 6085 | 5305 | 5745 |
| 2 | 5320 | 5375 | 5345 | 6485 | 5425 | 6025 | 5540 | **185** | **250** | 4865 | 5970 | 5925 | 5655 | 5340 | 5345 | 5440 |
| 3 | 5965 | 5660 | 6085 | 6690 | 5855 | 6105 | 5830 | **40** | **10** | 5690 | 6145 | 5760 | 5965 | 6220 | 5955 | 5845 |
| 4 | 5660 | 5920 | 6010 | 7105 | 5430 | 6135 | 5960 | 5865 | 6315 | 5460 | 6010 | 6245 | 5495 | 6085 | 5195 | 5790 |
| 5 | 6050 | 5635 | 5885 | 7055 | 5560 | 5950 | 6320 | 6030 | 6610 | 5405 | 6025 | 6185 | 6045 | 6005 | 5135 | 5855 |
| 6 | 5310 | 5435 | 5505 | **70** | 6000 | 6325 | 5735 | 5795 | **70** | 5435 | 5895 | 5685 | 5940 | 6025 | 5475 | 6035 |
| 7 | 5265 | 5620 | 5820 | **75** | 5990 | 5680 | 6210 | 5820 | **65** | 4995 | 5870 | 6100 | 6015 | 5945 | 5405 | 5875 |
| 8 | 6010 | 5945 | 5940 | 6515 | 5790 | 5625 | 6050 | 5965 | 6455 | 5490 | 5800 | 6135 | 6140 | 6160 | 5490 | 6095 |
| 9 | 5800 | 5810 | 6250 | 6410 | 5660 | 5905 | 6095 | 5740 | 6520 | 5615 | 5720 | 6025 | 5975 | 5860 | 5640 | 6250 |
| 10 | 4790 | 5240 | 5405 | 6825 | 5720 | 5605 | 6280 | **85** | **70** | 5210 | 6050 | 5960 | 5825 | **195** | 4375 | 5135 |
| 11 | 5790 | 5660 | 5770 | 6420 | 5805 | 6100 | 5985 | **85** | **85** | 5665 | 5945 | 6020 | 5505 | **220** | 4715 | 6080 |
| 12 | 5395 | 5075 | 5980 | 6420 | 5570 | 5905 | 5820 | 5950 | 6480 | 5660 | 6060 | 5735 | 5815 | 6260 | 4505 | 5345 |
| 13 | 5485 | 5320 | 5680 | 5970 | 5590 | 5745 | 5905 | 5995 | 6470 | 5365 | 5630 | 5905 | 5690 | 6165 | 5200 | 5250 |
| 14 | 5430 | 5500 | 5315 | 5715 | 5465 | 5780 | 5830 | 6430 | 6630 | 5325 | 5785 | 5730 | 5470 | 5620 | 5230 | 5755 |
| 15 | 5340 | 5820 | 5575 | 6095 | 5620 | 5800 | 6040 | 5645 | 6285 | 5135 | 5865 | 5870 | 5610 | 5865 | 5200 | 5715 |

Fig. 6.2: Impact of cross-layer overhead on data communication.

number of injected flits for most nodes is greater than 5,000. This number is less by orders of magnitude for software routing nodes (dark grey). Because of the large number of packets that have to be routed in software, these nodes are overloaded and are not able to inject own data flits.

The same can be observed for the intermediate destination nodes used for the shutdown of links in x-dimension, i.e.:

- nodes (7,2) & (8,2) for the link between nodes (7,3) & (8,3) and
- nodes (7,10) & (8,10) for the link between (7,11) & (8,11) (cf. *intermediate destination node (X)* in Figure 6.2).

Each packet that is redirected because of an x-dimension link shutdown is consumed and reinjected by one of the two corresponding intermediate destination nodes. Thus, these nodes are overloaded by cross-layer communication overhead as well.

None of the intermediate destination nodes used for the shutdown of a link in y-dimension, can be identified that has a considerably smaller number of injected flits. The reason for this is a more evenly distribution of the load of redirected packets, which is shared by all the nodes in a neighbored column (cf. *intermediate destination node (Y)* in Figure 6.2). In contrast to an x-dimension shutdown, the choice of selecting an intermediate destination node depends on the packet's destination. For instance, all packets for node (3,0) are redirected at (3,7) to the intermediate destination node (4,0); packets for (3,1) are redirected to (4,1) and so on. Thus, the cross-layer communication overhead for an intermediate destination node for a y-dimension link shutdown is less than for one of an x-dimension link shutdown.

In contrast to SBR, the cross-layer overhead of RHR does not include any data flits but is composed of flits used to communicate

- the availability information to the nodes within a unit,
- the ETG from a unit manager to the nodes, and
- the routing table entries to the RCU.

The measured cross-layer overhead in level 1 units resulting from the shutdown of communication resources is shown in Table 6.3 for RHR. If a link connecting two level 1 units is shut down, the cross-layer overhead is induced in both units.

The results show that RHR causes only a small cross-layer overhead of 2353 flits at maximum for five shutdowns. At the same time, approximately 10 million data flits are exchanged between layers for normal data communication. Thus, the overhead amounts to only 0.02%. Because of this small

Table 6.3: Average Cross-Layer Overhead RHR [flits]

| | Shutdowns | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| Availability Information | 89 | 134 | 191 | 248 | 317 |
| ETG | 300 | 435 | 627 | 800 | 1018 |
| Routing Table Entries | 300 | 435 | 627 | 800 | 1018 |
| Total Overhead | 691 | 1004 | 1445 | 1848 | 2353 |

overhead, none of the nodes is overloaded as in case of SBR, and thus, the data communication is not affected by the overhead.

During reconfiguration, data communication, however, is affected because all switches of a unit being reconfigured are in maintenance mode and do not inject data flits. (cf. Subsection 5.3.3.1). For the considered hierarchical network configuration (4x2,4x4) the reconfiguration time of a level 1 unit is about 17,000 cycles (cf. Figure 5.16). For the injection rate of 0.065 flits/node/cycle approximately every 16 cycles a flit is injected by every network node. This implies that every node of a unit under reconfiguration cannot inject about 1062 flits while being in maintenance mode. In case of level 2 reconfiguration (27,000 cycles), 1687 flits cannot be injected.

## *6.1.2 Performance of Cross-Layer Diagnosis*

### 6.1.2.1 Data Throughput

During the localization process of a permanent fault, diagnosis techniques introduce additional test data to the NoC, which occupy communication resources. Additionally, switches may become temporarily unavailable for data communication as in case of FD and SD techniques. Both result in a reduction in system performance of the NoC which is reflected by a decreased data throughput.

In this evaluation, the system performance of the cross-layer and standalone diagnosis techniques is investigated. For this purpose, the NoC is simulated using sets with one to five permanent faults. When a permanent fault is diagnosed by a technique, the corresponding faulty component is shut down.

To sustain data communication in the NoC after the component's shutdown, either SBR or RHR method is used.

The measured data throughput of the diagnosis techniques in combination with SBR is shown in Figure 6.3. The results show that the throughput de-

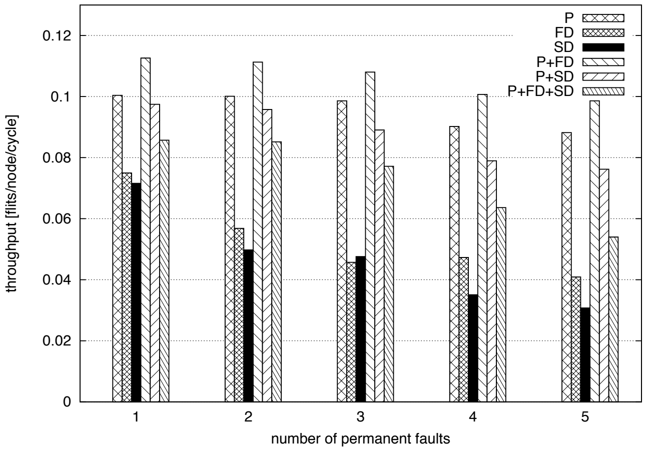


Fig. 6.3: Average data throughput for the different diagnosis techniques in combination with SBR.

creases for all diagnosis techniques with increasing number of faults. The reason for this is the increasing number of initiated diagnoses and the resulting communication resource shutdowns. It can also be observed that the achievable data throughput in principle is related to the diagnosis quality of the techniques (cf. Table 4.4): the higher the quality level of a technique, the lower the throughput. The only exception to this principle is the higher throughput value of P+FD compared to P. The reason is explained further below.

Comparing the standalone techniques P, FD, and SD, it can be observed that the achievable data throughput of FD and SD is much smaller than the one of P for every number of faults. While for five permanent faults the throughput of P is about 76% compared to the throughput of the fault-free case (0.115 flits/node/cycle), for FD and SD the achievable throughput is already reduced to 65% and 62% for one fault. For five permanent faults, the data throughput is reduced to approximately 35% (FD) and 26% (SD). The higher throughput of P compared to FD and SD can be attributed to P's smaller localization latency (cf. Table 4.5). Furthermore, while P only diagnoses those switches situated on a path, in case of FD and SD all switches are diagnosed, and thus, the data communication in the whole NoC is affected.

The results for the combined techniques (cf. Figure 6.3) show that all of them have a higher data throughput compared to the standalone techniques FD and SD. The reason for this is the reduction of set $\Phi$ of potentially faulty switches (cf. Listing 1) to one single switch by P in the first place. Thus, the subsequent FD or SD diagnosis has only to be carried out for the single switch instead of all NoC switches. By this means, the negative impact of diagnosis on the data communication is greatly reduced, as all other switches continue normal data communication. Of all combined techniques, P+FD+SD has the lowest throughput because of its higher localization latency compared to P+FD and P+SD (cf. Table 4.5).

When comparing the results of P and P+FD, it can be observed that P+FD, despite of its longer localization latency, has a higher data throughput than P. This higher throughput can be attributed to FD's capability to identify false positives reported by P, and thus, to prevent the shutdown of the respective communication resource. In case of P, a communication resource being erroneously diagnosed as faulty is always shut down. However, with the increasing number of permanent faults the throughput of P+FD is reduced faster than in case of P. Eventually, for five faults, the throughput of P+FD is less than the one of P. The reason for this is the increasing number of FD activations because of false positives reported by P.

The simulations have been repeated with the identical fault sets for the combination of diagnosis techniques with the RHR fault tolerance method. The corresponding data throughput results are shown in Figure 6.4. The results of the combination with RHR show a similar behavior as for the combination with SBR. Again, the cross-layer diagnosis techniques outperform FD and SD with respect to the data throughput. In contrast to the results for the combination with SBR, here, P has a comparable or higher throughput than P+FD. The reason for this is a reduced number of diagnosed false positives compared to the combination with SBR. On the one hand, this can be attributed to fact that packets that cannot be forwarded because of reconfiguration are discarded, and thus, the network load is reduced. On the other hand, most discarded packets can be delivered to their destination after the first retransmission, and thus, P is not activated for these packets, as the number of attempts to send these packets does not exceed the maximum of three. The reason for this is that a retransmission is triggered after a timeout $t_\delta$ of 40 kcycles. The required time for routing reconfiguration within a hierarchical unit on level $h = 1$ is about 28

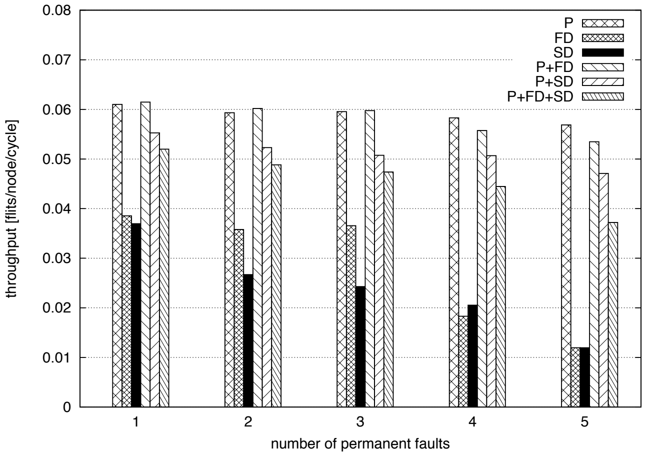


Fig. 6.4: Average data throughput for the different diagnosis techniques in combination with RHR.

kcycles in case of (4x2,4x4) (cf. Table 5.16). Thus, reconfiguration is already completed when a packet is retransmitted and for the communication of the packet the new routing is used.

### 6.1.2.2 Pareto Analysis

As shown in the last subsection, all cross-layer diagnosis techniques have a significantly higher data throughput than the lower layer standalone techniques FD and SD. However, as shown by the results in Subsection 4.4.2.2 (cf. Table 4.4 ), FD and SD techniques have a higher diagnosis quality than the cross-layer techniques. This shows that both design goals, performance and quality, are in conflict with each other. Thus, a diagnosis technique cannot offer best performance and best diagnosis quality at the same time but offers a tradeoff between both.

To identify those diagnosis techniques that offer an optimal tradeoff between data throughput ($tp$) and diagnosis quality ($dq$), a pareto analysis is performed for the techniques. A technique $T_1$ offers an optimal tradeoff, i.e. $T_1$ is pareto-optimal, if there exists no other technique $T_2$ that dominates $T_1$ regarding throughput and quality:

$$!\exists T_2 : T_2 \neq T_1 \wedge tp_{T_2} \geq tp_{T_1} \wedge dq_{T_2} \geq dq_{T_1} \tag{6.1}$$

The design space spanned by throughput and quality for the combination of diagnosis techniques with SBR is shown in Figure 6.5. The unachievable optimal solution, best performance and best quality, is situated in the upper right corner of the design space. The arrow leading from a white to a black symbol shows the throughput trend from one fault to five faults. For the sake of a better differentiation between techniques with overlapping trendlines, square and circle symbols are used.

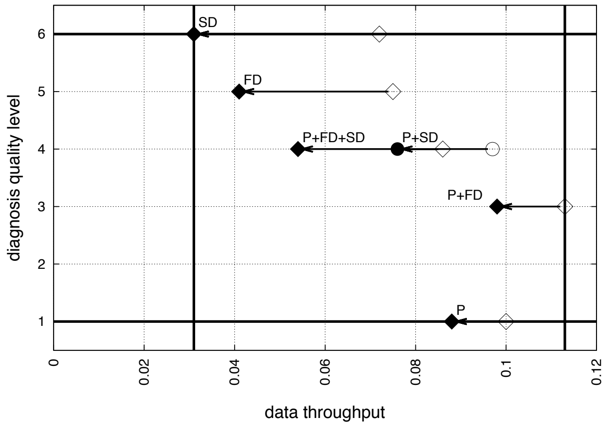


Fig. 6.5: Diagnosis quality vs. data throughput SBR.

For one to five permanent faults, the two cross-layer techniques P+FD and P+SD as well as the standalone techniques FD and SD are pareto-optimal. Technique P+FD+SD has the same diagnosis quality as P+SD, however, its throughput is less because of the higher time required to locate a fault. P is not a pareto-optimal solution either. Although P has the smallest diagnosis time required of all techniques (cf. Table 4.5), its throughput is less than the one of P+FD. This is because of the shutdown of communication resources erroneously diagnosed as faulty because of the missing feedback from one of the lower-layer diagnosis techniques.

The same design space for the combination of diagnosis techniques with RHR is shown in Figure 6.6. In contrast to the combination with SBR, for the

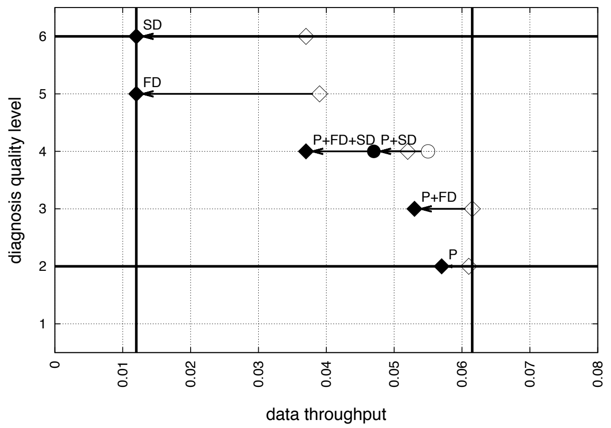


Fig. 6.6: Diagnosis quality vs. data throughput RHR.

combination with RHR, the number of diagnosed false positives is less and P's quality is comparable with $P_{mod}$ in Table 4.4.

For one permanent fault the same techniques are pareto-optimal as in case of SBR. For five permanent faults, however, FD is no longer pareto-optimal as SD has the same throughput while having a higher quality. On the other hand, P now belongs to the set of pareto-optimal solutions as it features the highest throughput.

In both design spaces it can be observed that the combinations P+FD and P+SD offer a good tradeoff between quality and performance. While both techniques have a better performance than FD and SD because of the reduction of set $\Phi$ by P, the feedback provided from FD or SD to P results in an increased diagnosis quality.

## 6.2 Summary

In summary, the results for the two fault tolerance methods SBR and RHR show that the exchange of information and data between software layer and other layers has an impact on system performance because of the additional load on the local communication link between switch and PE. If this load increases too much, as shown for SBR, the capability of a network node to communicate its own data is negatively affected.

The diagnosis evaluation results show that, regardless of the fault tolerance method used, all combinations of protocol P with FD or SD offer a smaller performance impact than the standalone techniques FD and SD. Due to the reduction of set $\Phi$ by P, the fault localization effort of the combined techniques is less than for the standalone techniques. The results with SBR further show that the increased diagnosis quality of the combined techniques compared to P helps to avoid performance loss due to the shutdown of resources erroneously diagnosed as faulty. The pareto analyses show that the cross-layer diagnosis techniques P+FD and P+SD are pareto-optimal and constitute a good tradeoff between performance and diagnosis quality.

# Chapter 7
# Conclusion and Future Work

In this dissertation, cross-layer interaction for fault diagnosis and fault tolerance was studied.

Cross-layer fault diagnosis was presented in Chapter 4. A general interaction scheme was proposed that enables the combination of protocol-based diagnosis of the transport layer with functional diagnosis and structural diagnosis of the network layer and the physical layer. A top-down information flow is used to locate faults, while a bottom-up flow provides information about diagnosis results of the lower layer techniques to the transport layer diagnosis.

For fault diagnosis on the transport layer, an end-to-end protocol was proposed. It consists of a base protocol for controlling end-to-end communication and for handling transient faults and a diagnosis protocol for localizing a faulty communication resource in a NoC. It was shown that by incorporating information about the network layer routing and topological information, the diagnosis protocol is able to narrow down a fault's position on a communication path to a single link, switch, or crossbar connection.

The protocol was combined with existing functional [27] and structural diagnosis [25] techniques according to the proposed interaction scheme. All combinations were evaluated regarding their diagnosis quality and the required fault localization latency. The results have shown that the combinations offer a tradeoff between quality and latency. Because the time-consuming step to identify a faulty switch with functional diagnosis and structural diagnosis is replaced by the much more efficient localization process of the protocol, the combinations have a considerably reduced fault localization latency compared to the standalone techniques. It was shown, however, that the latency reduction comes at the expense of a reduced diagnosis quality compared to standalone functional and structural diagnosis because of the less detailed information available on the transport layer about the NoC's underlying hardware. For the protocol, the combination constitutes an improvement in diagnosis quality compared to the standalone technique. The reason for this is the feedback from the lower layer techniques that prevents the shutdown of communication resources erroneously diagnosed as faulty (false positives) by the protocol.

As possible future work for cross-layer fault diagnosis, it can be studied how the diagnosis quality of the combined diagnosis techniques can be improved. As a starting point, the quality of transport layer diagnosis can be increased. For this purpose, investigations can be carried out to identify additional lower layer hardware information that helps to increase the quality. It must be taken into account, however, that the effort to gain the information and to provide it to the transport layer does not impair the performance benefit of the combined diagnosis techniques. Furthermore, the number of reported false positives can be reduced using the existing information. As a side benefit of reducing the number of false positives, the number of lower layer diagnosis executions is reduced as well, which in turn reduces the impact on NoC performance.

Chapter 5 dealt with cross-layer fault tolerance. Two approaches were presented that make use of software methods to provide fault tolerance for network layer routing. At first, the requirements for providing information of lower layers to the software using standard data communication resources and interfaces were discussed. The concepts of two data link layer mechanisms were presented that, on the one hand, transform status information into communicable units and vice versa and, on the other hand, take care that the communication resources are not blocked.

As the first of the two fault tolerance approaches, software-based packet rerouting was proposed. Packets that cannot be forwarded by a node's switch are sent to the processing element instead. It was shown that by incorporating information about the network layer routing and topological information, it is possible to reroute packets to an alternative path via an intermediate destination in software, and thus, to create fault tolerance for otherwise not fault tolerant deterministic network layer routings. An implementation of the software rerouting algorithm for dimension order XY routing was presented and was used for evaluation. The evaluation results have shown that rerouting packets in software is suitable for NoCs with small or moderate network load. For high load, many rerouted packets had to be discarded and retransmitted at a later point in time because of full software buffers of software routing and intermediate destination nodes resulting in a performance loss of the NoC.

As future work, investigation can be made to relieve nodes of the load caused by the rerouting of packets. One possibility is that the node rerouting a packet destined for a receiver informs the packet's sender about the unavailable link. With this information it will become possible for the sender to calculate an intermediate destination node for all packets for the same receiver and to redirect the packets directly to this node. This distributes calculation effort and load in the network more evenly. One challenge, however, is to prevent all informed senders from redirecting their packets via the same intermediate destination node, since otherwise this node will become overloaded instead.

As the second approach, a reconfigurable routing for hierarchically organized NoC topologies was proposed. It was shown that by means of cross-layer interaction the autonomous management of hierarchical units as well as the autonomous calculation of routing within the units become possible. First, the hierarchical network concept was discussed and the basic routing principles were presented. Following this, the abstract topology graph was introduced and the algorithms used to calculate the routing using this graph were shown. A cross-layer reconfiguration process for the hierarchical routing was presented, consisting of a combination of software and hardware methods that is divided into three phases.

Evaluation has shown that the cross-layer reconfiguration process requires more time to finish compared to state-of-the-art reconfiguration on the network layer in case of non-hierarchical topologies. This was mainly caused by the time required by the software algorithm to recalculate the routing. However, by introducing a logical hierarchy and unit autonomy, it was shown that the routing performance can be increased and that the reconfiguration time can be reduced significantly compared to state-of-the-art network layer reconfiguration.

The failure of a manager node implies the loss of the complete management of a unit. System operation may continue unaffected as long as no permanent fault occurs in this unit. However, in case of a permanent fault, routing reconfiguration is no longer possible. In the worst case, this results in a complete system failure. For this reason, investigations can be made to cope with the failure of a manager, e.g. by migrating the manager's functionality to other nodes. As a starting point, the integer linear programming optimization presented in the works [120] and [142] can be used to determine spare managers. As an alternative solution, the affected hierarchical unit can be dissolved and its nodes can be reassigned to neighbored units.

Finally, in Chapter 6, the diagnosis techniques were combined with the fault tolerance approaches and the combinations were evaluated. It was shown that intense cross-layer information exchange with the software may cause a considerable reduction of NoC performance. As a consequence, future cross-layer designs have to pay attention to minimize the amount of information communicated from lower layers to the software and vice versa in order not to block resources required for data communication. The results have further confirmed that cross-layer design is able to reduce the diagnosis effort and the impact of diagnosis on NoC performance. The pareto analyses for packet rerouting and for hierarchical routing have shown that the combinations of protocol-based diagnosis with functional diagnosis and protocol-based diagnosis with structural diagnosis are pareto-optimal and offer a good tradeoff between performance and diagnosis quality in both cases.

In conclusion, the evaluations in this dissertation have shown that cross-layer design always constitutes a tradeoff between different parameters. While cross-layer interaction provides a wider scope of information that helps to deal with faults, the effort to gather this information and the communication load across layers increases. Future work on cross-layer design will have to further investigate the interaction of layers and the tradeoffs emerging from cross-layer interaction with respect to the given field of application to exploit the full potential.

# References

[1] Rawan Abdel-Khalek and Valeria Bertacco. “Post-silicon Platform for the Functional Diagnosis and Debug of Networks-on-chip”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 13.3s (Mar. 2014), pp. 1–25. ISSN: 1539-9087. DOI: 10.1145/2567936.

[2] Muhammad Afzal. “Design and Implementation of a Fault Tolerant VHDL Switch with Reconfigurable Routing Tables”. Master Thesis (MSc). Germany: University of Stuttgart, 2013.

[3] Ibrahim Ahmed. “Reliable Routing Table Reconfiguration for On-chip Network Switches”. Master Thesis (MSc). Germany: University of Stuttgart, 2014.

[4] K. Aisopos, A. DeOrio, Li-Shiuan Peh, and V. Bertacco. “ARIADNE: Agnostic Reconfiguration in a Disconnected Network Environment”. In: *Proc. of IEEE/ACM Int’l Conf. on Parallel Architectures and Compilation Techniques (PACT)*. Oct. 2011, pp. 298–309. DOI: 10.1109/PACT.2011.61.

[5] Muhammad Ali, Michael Welzl, Sven Hessler, and Sybille Hellebrand. “An efficient fault tolerant mechanism to deal with permanent and transient failures in a network on chip”. In: *Int. J. High Performance System Architecture (IJHPSA)* 1.2 (2007), pp. 113–123. ISSN: 1751-6528. DOI: 10.1504/IJHPSA.2007.015397.

[6] Alexandre Amory, Frederico Ferlini, Marcelo Lubaszewski, and Fernando Moraes. “DfT for the Reuse of Networks-on-Chip as Test Access Mechanism”. In: *Proc. of the 25th VLSI Test Symposium (VTS)*. June 2007, pp. 435–440. DOI: 10.1109/VTS.2007.26.

[7] A.M. Amory et al. “A scalable test strategy for network-on-chip routers”. In: *Proc. of IEEE Int’l Test Conf. (ITC)*. Nov. 2005, pp. 590–599. DOI: 10.1109/TEST.2005.1584020.

[8] Nikolaos Batzolis. “Fault Tolerant End-to-End Flow Control Protocol for Networks on Chip”. Master Thesis (MSc). Germany: University of Stuttgart, 2011. URL: ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/.MSTR-3197/MSTR-3197.pdf.

[9] L. Bauer et al. “Adaptive Multi-Layer Techniques for Increased System Dependability”. In: *it - Information Technology* 57.3 (June 2015), pp. 149–158. ISSN: 2196-7032. DOI: 10.1515/itit-2014-1082.

[10] Robert Baumann. “Soft errors in advanced computer systems”. In: *Design Test of Computers, IEEE* 22.3 (May 2005), pp. 258–266. ISSN: 0740-7475. DOI: 10.1109/MDT.2005.69.

[11] Richard Bellman. “On a Routing Problem”. In: *Quarterly of Applied Mathematics* 16 (1958), pp. 87–90.

[12] Luca Benini and Giovanni De Micheli. “Networks on Chips: A New SoC Paradigm”. In: *Computer* 35.1 (Jan. 2002), pp. 70–78. ISSN: 0018-9162. DOI: 10.1109/2.976921.

[13] S. Borkar. "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation". In: *IEEE Micro* 25.6 (2005), pp. 10–16. ISSN: 0272-1732. DOI: 10.1109/MM.2005.110.

[14] Shekhar Borkar. "Design Perspectives on 22nm CMOS and Beyond". In: *Proc. of the 46th Annual Design Automation Conference (DAC)*. San Francisco, CA, USA: ACM, 2009, pp. 93–94. ISBN: 978-1-60558-497-3.

[15] Shekhar Borkar et al. "Parameter Variations and Impact on Circuits and Microarchitecture". In: *Proc. of the 40th Annual Design Automation Conference (DAC)*. Anaheim, CA, USA: ACM, 2003, pp. 338–342. ISBN: 1-58113-688-9. DOI: 10.1145/775832.775920.

[16] A. Burns, J. Harbin, and L. S. Indrusiak. "A Wormhole NoC Protocol for Mixed Criticality Systems". In: *IEEE Real-Time Systems Symposium (RTSS)*. Dec. 2014, pp. 184–195. DOI: 10.1109/RTSS.2014.13.

[17] S. Carrillo et al. "Scalable Hierarchical Network-on-Chip Architecture for Spiking Neural Network Hardware Implementations". In: *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 24.12 (2013), pp. 2451–2461. ISSN: 1045-9219. DOI: 10.1109/TPDS.2012.289.

[18] Nicholas P. Carter, Helia Naeimi, and Donald S. Gardner. "Design Techniques for Cross-layer Resilience". In: *Proc of the Conf. on Design, Automation and Test in Europe (DATE)*. Dresden, Germany, 2010, pp. 1023–1028. ISBN: 978-3-9810801-6-2.

[19] N. Caselli, A. Strano, D. Ludovici, and D. Bertozzi. "Cooperative Built-in Self-Testing and Self-Diagnosis of NoC Bisynchronous Channels". In: *Proc. of 6th IEEE Int'l Symp. on Embedded Multicore Socs (MCSoC)*. Sept. 2012, pp. 159–166. DOI: 10.1109/MCSoC.2012.13.

[20] Nicola Concer et al. "CTC: An end-to-end flow control protocol for multi-core systems-on-chip". In: Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 193–202. ISBN: 978-1-4244-4142-6. DOI: 10.1109/NOCS.2009.5071467.

[21] Christian Constantinescu. "Intermittent faults and effects on reliability of integrated circuits". In: *Proc. of Annual Reliability and Maintainability Symp. (RAMS)*. Jan. 2008, pp. 370–374. DOI: 10.1109/RAMS.2008.4925824.

[22] Alejandro Cook, Melanie Elm, Hans-Joachim Wunderlich, and Ulrich Abelein. "Structural In-Field Diagnosis for Random Logic Circuits". In: *Proc. of the 16th IEEE European Test Symp. (ETS)*. 2011, pp. 111–116. DOI: 10.1109/ETS.2011.25.

[23] Erika Cota, Alexandre de Morais Amory, and Marcelo Soares Lubaszewski. *Reliability, Availability and Serviceability of Networks-on-Chip*. 1st. Springer, 2012. ISBN: 9781461407904.

[24] M. Dadashi, L. Rashid, K. Pattabiraman, and S. Gopalakrishnan. "Hardware-Software Integrated Diagnosis for Intermittent Hardware Faults". In: *Proc. of 44th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN)*. June 2014, pp. 363–374. DOI: 10.1109/DSN.2014.1.

[25] A. Dalirsani, S. Holst, M. Elm, and H. Wunderlich. "Structural Test for Graceful Degradation of NoC Switches". In: *Proc. of 16th IEEE European Test Symp. (ETS)*. 2011, pp. 183–188. DOI: 10.1109/ETS.2011.33.

[26] A. Dalirsani, M.E. Imhof, and H.-J. Wunderlich. “Structural Software-Based Self-Test of Network-on-Chip”. In: *Proc. of 32nd IEEE VLSI Test Symp. (VTS)*. Apr. 2014, pp. 1–6. DOI: 10.1109/VTS.2014.6818754.

[27] A. Dalirsani et al. “On Covering Structural Defects in NoCs by Functional Tests”. In: *Proc. of 23rd IEEE Asian Test Symp. (ATS)*. Nov. 2014, pp. 87–92. DOI: 10.1109/ATS.2014.27.

[28] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., 2003. ISBN: 0122007514.

[29] W.J. Dally. “Virtual-channel flow control”. In: *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 3.2 (1992), pp. 194–205. ISSN: 1045-9219. DOI: 10.1109/71.127260.

[30] W.J. Dally and C.L. Seitz. “Deadlock-Free Message Routing in Multiprocessor Interconnection Networks”. In: *IEEE Transactions on Computers (TC)* C-36.5 (1987), pp. 547–553. ISSN: 0018-9340. DOI: 10.1109/TC.1987.1676939.

[31] W.J. Dally and B. Towles. “Route packets, not wires: on-chip interconnection networks”. In: *Proc. of Design Automation Conf. (DAC)*. 2001, pp. 684–689. DOI: 10.1109/DAC.2001.156225.

[32] Dana-Elena Damaschin. “Fault Tolerant Management of Communication Channels of a NoC Switch”. Master Thesis (MSc). Germany: University of Stuttgart, 2014.

[33] R. Das et al. “Design and evaluation of a hierarchical on-chip interconnect for next-generation CMPs”. In: *Proc. of 15th IEEE Int’l Symp. on High Performance Computer Architecture (HPCA)*. 2009, pp. 175–186. DOI: 10.1109/HPCA.2009.4798252.

[34] Giovanni De Micheli and Luca Benini. *Network On Chips*. 1st. Morgan Kaufmann, 2006. ISBN: 978-0-12-370521-1.

[35] Sujay Deb and Hemanta Kumar Mondal. “Wireless network-on-chip: a new era in multi-core chip design”. In: *Proc. of 25nd IEEE International Symposium on Rapid System Prototyping (RSP)*. Oct. 2014, pp. 59–64. DOI: 10.1109/RSP.2014.6966893.

[36] Sujay Deb et al. “Wireless NoC as Interconnection Backbone for Multicore Chips: Promises and Challenges”. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)* 2.2 (June 2012), pp. 228–239. DOI: 10.1109/JETCAS.2012.2193835.

[37] A. DeHon, H. M. Quinn, and N. P. Carter. “Vision for Cross-Layer Optimization to Address the Dual Challenges of Energy and Reliability”. In: *Proc of Design, Automation Test in Europe Conference Exhibition (DATE)*. Mar. 2010, pp. 1017–1022. DOI: 10.1109/DATE.2010.5456959.

[38] M. Dehyadgari, M. Nickray, A. Afzali-Kusha, and Z. Navabi. “A New Protocol Stack Model for Network on Chip”. In: *Proc. of IEEE Computer Society Annual Symp. on Emerging VLSI Technologies and Architectures (ISVLSI)*. Mar. 2006, pp. 440–441. ISBN: 0-7695-2533-4. DOI: 10.1109/ISVLSI.2006.7.

[39] Onur Derin, Deniz Kabakci, and Leandro Fiorin. “Online Task Remapping Strategies for Fault-tolerant Network-on-Chip Multiprocessors”. In: *Proc. of the Fifth ACM/IEEE Int’l Symp. on Networks-on-Chip (NOCS)*. Pittsburgh, Pennsylvania: ACM, 2011, pp. 129–136. ISBN: 978-1-4503-0720-8. DOI: 10.1145/1999946.1999967.

[40] Edsger W. Dijkstra. “A Note on Two Problems in Connexion with Graphs”. In: *Numerische Mathematik* 1.1 (Dec. 1959), pp. 269–271. ISSN: 0029-599X. DOI: 10.1007/BF01386390.

[41] Masoumeh Ebrahimi and Masoud Daneshtalab. “A Light-weight fault-tolerant routing algorithm tolerating faulty links and routers”. In: *Computing* 97.6 (2013), pp. 631–648. ISSN: 1436-5057. DOI: 10.1007/s00607-013-0362-9.

[42] J.M. Emmert, C.E. Stroud, and M. Abramovici. “Online Fault Tolerance for FPGA Logic Blocks”. In: *IEEE Transactions on Very Large Scale Integration Systems (VLSI)* 15.2 (Feb. 2007), pp. 216–226. ISSN: 1063-8210. DOI: 10.1109/TVLSI.2007.891102.

[43] David Fick et al. “Vicis: a reliable network for unreliable silicon”. In: *DAC ’09: Proceedings of the 46th Annual Design Automation Conference*. San Francisco, California: ACM, 2009, pp. 812–817. ISBN: 978-1-60558-497-3. DOI: 10.1145/1629911.1630119.

[44] D. Fick et al. “A highly resilient routing algorithm for fault-tolerant NoCs”. In: *Proc of Design, Automation Test in Europe Conference Exhibition (DATE)*. Apr. 2009, pp. 21–26. DOI: 10.1109/DATE.2009.5090627.

[45] L. Fiorin and M. Sami. “Fault-Tolerant Network Interfaces for Networks-on-Chip”. In: *IEEE Transactions on Dependable and Secure Computing (TDSC)* 11.1 (Jan. 2014), pp. 16–29. ISSN: 1545-5971. DOI: 10.1109/TDSC.2013.28.

[46] Robert W. Floyd. “Algorithm 97: Shortest Path”. In: *Commun. ACM* 5.6 (June 1962), pp. 345–. ISSN: 0001-0782. DOI: 10.1145/367766.368168.

[47] A.P. Frantz et al. “Crosstalk- and SEU-Aware Networks on Chips”. In: *Design Test of Computers, IEEE* 24.4 (July 2007), pp. 340–350. ISSN: 0740-7475. DOI: 10.1109/MDT.2007.128.

[48] Y. Fukushima, M. Fukushi, and S. Horiguchi. “Fault-Tolerant Routing Algorithm for Network on Chip without Virtual Channels”. In: *Proc. of 24th IEEE Int’l Symp. on Defect and Fault Tolerance in VLSI Systems (DFT)*. Oct. 2009, pp. 313–321. DOI: 10.1109/DFT.2009.41.

[49] A Garbade et al. “Fault Localization in NoCs Exploiting Periodic Heartbeat Messages in a Many-Core Environment”. In: *Proc. of 27th IEEE Int’l Parallel and Distributed Processing Symp. Workshops & PhD Forum (IPDPSW)*. 2013, pp. 791–795. DOI: 10.1109/IPDPSW.2013.150.

[50] Fayez Gebali, Haytham Elmiligi, and Mohamed Watheq El-Kharashi. *Networks-on-Chips Theory and Practice*. 1st. CRC Press, 2009. ISBN: 9781420079784.

[51] Alberto Ghiribaldi, Alessandro Strano, Michele Favalli, and Davide Bertozzi. “Power efficiency of switch architecture extensions for fault tolerant NoC design”. In: *Proc. of Int’l Green Computing Conference (IGCC)*. June 2012, pp. 1–6. DOI: 10.1109/IGCC.2012.6322281.

[52] A. Ghofrani et al. “Comprehensive online defect diagnosis in on-chip networks”. In: *Proc. of 30th IEEE VLSI Test Symp. (VTS)*. 2012, pp. 44–49. DOI: 10.1109/VTS.2012.6231078.

[53] P. Ghosh, A. Sen, and A. Hall. “Energy efficient application mapping to NoC processing elements operating at multiple voltage levels”. In: *Proc. of 3rd ACM/IEEE Int’l Symp. on Networks-on-Chip (NoCS)*. May 2009, pp. 80–85. DOI: 10.1109/NOCS.2009.5071448.

[54] B. Ghoshal, K. Manna, S. Chattopadhyay, and I. Sengupta. “In-Field Test for Permanent Faults in FIFO Buffers of NoC Routers”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.1 (Jan. 2016), pp. 393–397. ISSN: 1063-8210. DOI: 10.1109/TVLSI.2015.2393714.

[55] J. Gracia-Moran et al. “Defining a Representative and Low Cost Fault Model Set for Intermittent Faults in Microprocessor Buses”. In: *Proc. of Sixth Latin-American Symp. on Dependable Computing (LADC)*. Apr. 2013, pp. 98–103. DOI: 10.1109/LADC.2013.19.

[56] C. Grecu, P. Pande, A. Ivanov, and R. Saleh. “BIST for network-on-chip interconnect infrastructures”. In: *Proc. of 24th IEEE VLSI Test Symp. (VTS)*. 2006, pages. DOI: 10.1109/VTS.2006.22.

[57] Pierre Guerrier and Alain Greiner. “A Generic Architecture for On-chip Packet-switched Interconnections”. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. DATE ’00. Paris, France: ACM, 2000, pp. 250–256. ISBN: 1-58113-244-1. DOI: 10.1145/343647.343776.

[58] T. Han, I. Choi, H. Oh, and S. Kang. “A Scalable and Parallel Test Access Strategy for NoC-Based Multicore System”. In: *Proc. of IEEE 23rd Asian Test Symposium (ATS)*. Nov. 2014, pp. 81–86. DOI: 10.1109/ATS.2014.26.

[59] Marcos Herve, Erika Cota, Fernanda Lima Kastensmidt, and Marcelo Lubaszewski. “Diagnosis of interconnect shorts in mesh NoCs”. In: *Networks-on-Chip, International Symposium on* (2009), pp. 256–265. DOI: 10.1109/NOCS.2009.5071475.

[60] S. Holmbacka, W. Lund, S. Lafond, and J. Lilius. “Task Migration for Dynamic Power and Performance Characteristics on Many-Core Distributed Operating Systems”. In: *Proc. of 21st Euromicro Int’l Conf. on Parallel, Distributed, and Network-Based Processing (PDP)*. Feb. 2013, pp. 310–317. DOI: 10.1109/PDP.2013.52.

[61] R. Holsmark and S. Kumar. “An Abstraction to Support Design of Deadlock-free Routing Algorithms for Large and Hierarchical NoCs”. In: *Proc. of 11th IEEE Int’l Conf. on Computer and Information Technology (CIT)*. 8. 2011, pp. 59–66. DOI: 10.1109/CIT.2011.32.

[62] R. Holsmark, S. Kumar, M. Palesi, and A. Mejia. “HiRA: A methodology for deadlock free routing in hierarchical networks on chip”. In: *Proc. of 3rd ACM/IEEE Int’l Symp. on Networks-on-Chip (NOCS)*. 2009, pp. 2–11. DOI: 10.1109/NOCS.2009.5071439.

[63] M. Hosseinabady, A. Dalirsani, and Z. Navabi. “Using the Inter- and Intra-Switch Regularity in NoC Switch Testing”. In: *Proc. of Design, Automation & Test in Europe Conf. (DATE)*. 2007, pp. 1–6. DOI: 10.1109/DATE.2007.364618.

[64] HP Labs CACTI. visited May 2017. URL: http://www.hpl.hp.com/research/cacti/.

[65] L. Huang et al. “Non-Blocking Testing for Network-on-Chip”. In: *IEEE Transactions on Computers (TC)* 65.3 (Mar. 2016), pp. 679–692. ISSN: 0018-9340. DOI: 10.1109/TC.2015.2489216.

[66] Intel Xeon Phi Coprocessor. visited May 2017. URL: https://software.intel.com/de-de/mic-developer.

[67] International Technology Roadmap For Semiconductors. visited May 2017. URL: http://www.itrs2.net/itrs-reports.html.

[68] C. Iordanou, V. Soteriou, and K. Aisopos. “Hermes: Architecting a top-performing fault-tolerant routing algorithm for Networks-on-Chips”. In: *Proc. of IEEE 32nd Int’l Conf. on Computer Design (ICCD)*. Oct. 2014, pp. 424–431. DOI: 10.1109/ICCD.2014.6974715.

[69] Axel Jantsch and Hannu Tenhunen. *Network On Chips*. 1st. Kluwer Academic, 2003. ISBN: 1402073925.

[70] M.R. Kakoee, V. Bertacco, and L. Benini. “At-Speed Distributed Functional Testing to Detect Logic and Delay Faults in NoCs”. In: *IEEE Transactions on Computers (TC)* 63.3 (Mar. 2014), pp. 703–717. ISSN: 0018-9340. DOI: 10.1109/TC.2013.202.

[71] N. Karimi, A. Alaghi, M. Sedghi, and Z. Navabi. “Online Network-on-Chip Switch Fault Detection and Diagnosis Using Functional Switch Faults”. In: *Journal of Universal Computer Science (UCS)* 14.22 (2008), pp. 3716–3736.

[72] H. S. Kia and C. Ababei. “Improving Fault Tolerance of Network-on-Chip Links via Minimal Redundancy and Reconfiguration”. In: *Proc. of Int’l Conf. on Reconfigurable Computing and FPGAs (ReConFig)*. Nov. 2011, pp. 363–368. DOI: 10.1109/ReConFig.2011.52.

[73] Tim Kogel, Rainer Leupers, and Heinrich Meyr. *Integrated System-Level Modeling of Network-on-Chip enabled Multi-Processor Platforms*. 1st. Springer, 2006. ISBN: 9781402048262.

[74] A. Kohler, G. Schley, and M. Radetzki. “Fault Tolerant Network on Chip Switching With Graceful Performance Degradation”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 29.6 (June 2010), pp. 883–896. ISSN: 0278-0070. DOI: 10.1109/TCAD.2010.2048399.

[75] Adan Kohler, Juan Manuel Castillo-Sanchez, Joachim Gross, and Martin Radetzki. “Minimal MPI as Programming Interface for Multicore System-on-Chips”. In: *Proc. of IEEE Forum on Specification and Design Languages (FDL)*. 2012, pp. 127–134. ISBN: 978-1-4673-1240-0.

[76] Anelise Kologeski, Caroline Concatto, Fernanda Lima Kastensmidt, and Luigi Carro. “Fault-Tolerant Techniques to Manage Yield and Power Constraints in Network-on-Chip Interconnections”. In: *VLSI-SoC: From Algorithms to Circuits and System-on-Chip Design: 20th IFIP WG 10.5/IEEE Int’l Conf. on Very Large Scale Integration (VLSI-SoC), Revised Selected Papers*. Ed. by Andreas Burg et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 144–161. ISBN: 978-3-642-45073-0. DOI: 10.1007/978-3-642-45073-0_8.

[77] Israel Koren and Mani Krishna. *Fault-Tolerant Systems*. 1st. Morgan Kaufmann, 2007. ISBN: 9780120885251.

[78] Doowon Lee, Ritesh Parikh, and Valeria Bertacco. “Brisk and Limited-impact NoC Routing Reconfiguration”. In: *Proc. of the Conf. on Design, Automation & Test in Europe (DATE)*. Dresden, Germany, 2014, pp. 306–306. ISBN: 978-3-9815370-2-4.

[79] L. Leem et al. “Cross-layer error resilience for robust systems”. In: *Proc. of IEEE/ACM Int’l Conf. on Computer-Aided Design (ICCAD)*. Nov. 2010, pp. 177–180. DOI: 10.1109/ICCAD.2010.5654129.

[80] T. Lehtonen, P. Liljeberg, and J. Plosila. “Fault Tolerance Analysis of NoC Architectures”. In: *Proc. on Int’l Symp. on Circuits and Systems (ISCAS)*. May 2007, pp. 361–364. DOI: 10.1109/ISCAS.2007.378464.

[81] Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila. "Analysis of Forward Error Correction Methods for Nanoscale Networks-on-chip". In: *Proc. of the 2nd Int'l Conf. on Nano-Networks (Nano-Net)*. Catania, Italy: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007, 3:1–3:5. ISBN: 978-963-9799-10-3.

[82] T. Lehtonen et al. "Self-Adaptive System for Addressing Permanent Errors in On-Chip Interconnects". In: *IEEE Transactions on Very Large Scale Integration Systems (VLSI)* 18.4 (Apr. 2010), pp. 527–540. ISSN: 1063-8210. DOI: 10.1109/TVLSI.2009.2013711.

[83] Ming Li, Wen-Ben Jone, and Qing-An Zeng. "An efficient wrapper scan chain configuration method for network-on-chip testing". In: *in Proc. IEEE Computer Society Annual Symp. on Emerging VLSI Technologies and Architectures (ISVLSI)*. 2006, pp. 147–152. DOI: 10.1109/ISVLSI.2006.21.

[84] Z. Li et al. "Aurora: A Cross-Layer Solution for Thermally Resilient Photonic Network-on-Chip". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23.1 (Jan. 2015), pp. 170–183. ISSN: 1063-8210. DOI: 10.1109/TVLSI.2014.2300477.

[85] Jens Lienig. "Electromigration and Its Impact on Physical Design in Future Technologies". In: *Proc. of ACM Int'l Symp. on Physical Design (ISPD)*. Stateline, Nevada, USA, 2013, pp. 33–40. ISBN: 978-1-4503-1954-6. DOI: 10.1145/2451916.2451925.

[86] Yijun Liu, Zhenkun Li, Pinghua Chen, and Zhusong Liu. "A Stacked NoC Architecture for Quality-of-Service". In: vol. 1. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 609–612. ISBN: 978-0-7695-3494-7. DOI: 10.1109/ISISE.2008.241.

[87] Zhonghai Lu, Bei Yin, and A. Jantsch. "Connection-oriented multicasting in wormhole-switched networks on chip". In: *IEEE Computer Society Annual Symp. on Emerging VLSI Technologies and Architectures (ISVLSI)*. Mar. 2006, pages. DOI: 10.1109/ISVLSI.2006.31.

[88] O. Lysne, T.M. Pinkston, and J. Duato. "Part II: A Methodology for Developing Deadlock-Free Dynamic Network Reconfiguration Processes". In: *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 16.5 (2005), pp. 428–443.

[89] Kurt Mehlhorn and Peter Sanders. *Algorithms and Data Structures*. 1st. Springer, 2008. ISBN: 9783642096822.

[90] P. Meloni et al. "System Adaptivity and Fault-Tolerance in NoC-based MPSoCs: The MADNESS Project Approach". In: *Proc. of 15th Euromicro Conference on Digital System Design (DSD)*. Sept. 2012, pp. 517–524. DOI: 10.1109/DSD.2012.122.

[91] M. Millberg et al. "The Nostrum backbone-a communication protocol stack for Networks on Chip". In: *VLSI Design, 2004. Proceedings. 17th Int'l Conference on*. 2004, pp. 693–696. DOI: 10.1109/ICVD.2004.1261005.

[92] J.M. Montanana, J. Flich, and J. Duato. "Epoch-based reconfiguration: Fast, simple, and effective dynamic network reconfiguration". In: *Proc. of IEEE Int'l Symp. on Parallel and Distributed Processing (IPDPS)*. Apr. 2008, pp. 1–12. DOI: 10.1109/IPDPS.2008.4536298.

[93] Shubhendu S. Mukherjee, Joel Emer, and Steven K. Reinhardt. "The Soft Error Problem: An ArchitecturalPerspective". In: *Proc. of the 11th Int'l Symp. on High-*

*Performance Computer Architecture (HPCA)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 243–247. ISBN: 0-7695-2275-0. DOI: 10.1109/HPCA.2005.37.

[94] Srinivasan Murali. *Designing Reliable and Efficient Networks on Chips*. 1st. Springer Netherlands, 2009. ISBN: 9781402097560. DOI: 10.1007/978-4020-9757-7.

[95] Srinivasan Murali et al. "Analysis of Error Recovery Schemes for Networks on Chips". In: *IEEE Design and Test of Computers* 22.5 (2005), pp. 434–442. ISSN: 0740-7475. DOI: 10.1109/MDT.2005.104.

[96] S. Murali et al. "Designing Message-Dependent Deadlock Free Networks on Chips for Application-Specific Systems on Chips". In: *Proc. of Int'l Conf. on Very Large Scale Integration (IFIP)*. Oct. 2006, pp. 158–163. DOI: 10.1109/VLSISOC.2006.313226.

[97] L. M. Ni and P. K. McKinley. "A survey of wormhole routing techniques in direct networks". In: *Computer* 26.2 (Feb. 1993), pp. 62–76. ISSN: 0018-9162. DOI: 10.1109/2.191995.

[98] A. Nickelsen, J. Gronbaek, T. Renier, and H. P. Schwefel. "Probabilistic Network Fault-Diagnosis Using Cross-Layer Observations". In: *Proc. of Int'l Conf. on Advanced Information Networking and Applications (AINA)*. May 2009, pp. 225–232. DOI: 10.1109/AINA.2009.66.

[99] C. Nicopoulos et al. "On the Effects of Process Variation in Network-on-Chip Architectures". In: *IEEE Transactions on Dependable and Secure Computing (TDSC)* 7.3 (July 2008), pp. 240–254. ISSN: 1545-5971. DOI: 10.1109/TDSC.2008.59.

[100] Open Systems Interconnection Model (OSI). *ISO/IEC 7498-1:1994*. visited May 2017. URL: http://www.iso.org/.

[101] Michael Orshansky, Sani Nassif, and Duane Boning. *Design for Manufacturability and Statistical Design: A Constructive Approach*. 1st. Springer Publishing Company, Incorporated, 2010. ISBN: 9781441940445.

[102] Maurizio Palesi, Shashi Kumar, and Vincenzo Catania. "Leveraging Partially Faulty Links Usage for Enhancing Yield and Performance in Networks-on-Chip". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 29.3 (Mar. 2010), pp. 426–440. DOI: 10.1109/TCAD.2010.2041851.

[103] Ritesh Parikh and Valeria Bertacco. "Resource Conscious Diagnosis and Reconfiguration for NoC Permanent Faults". In: *IEEE Transactions on Computers (TC)* 65.7 (July 2016), pp. 2241–2256. ISSN: 0018-9340. DOI: 10.1109/TC.2015.2479586.

[104] Sudeep Pasricha and Nikil Dutt. *On-Chip Communication Architectures*. 1st. Morgan Kaufmann, 2008. ISBN: 9780123738929.

[105] C. Pinart. "A multilayer fault localization framework for IP over all-optical multilayer networks". In: *IEEE Network* 23.3 (May 2009), pp. 4–9. ISSN: 0890-8044. DOI: 10.1109/MNET.2009.4939257.

[106] M.K. Puthal, V. Singh, M.S. Gaur, and V. Laxmi. "C-Routing: An adaptive hierarchical NoC routing methodology". In: *Proc. of 19th IEEE/IFIP Int'l Conf. on VLSI and System-on-Chip (VLSI-SoC)*. 2011, pp. 392–397. DOI: 10.1109/VLSISoC.2011.6081616.

[107] C. Puttmann, J.-C. Niemann, M. Porrmann, and U. Ruckert. “GigaNoC - A Hierarchical Network-on-Chip for Scalable Chip-Multiprocessors”. In: *Proc. of 10th Euromicro Conf. on Digital System Design Architectures, Methods and Tools (DSD)*. 2007, pp. 495–502. DOI: `10.1109/DSD.2007.4341514`.

[108] Martin Radetzki, Chaochao Feng, Xueqian Zhao, and Axel Jantsch. “Methods for Fault Tolerance in Networks-on-chip”. In: *ACM Comput. Surv.* 46.1 (July 2013), 8:1–8:38. ISSN: 0360-0300. DOI: `10.1145/2522968.2522976`.

[109] Jaan Raik, Raimund Ubar, and Vineeth Govind. “Test Configurations for Diagnosing Faulty Links in NoC Switches”. In: *Proc. of the 12th IEEE European Test Symposium (ETS)*. May 2007, pp. 29–34. DOI: `10.1109/ETS.2007.41`.

[110] P. Ren et al. “A Deadlock-Free and Connectivity-Guaranteed Methodology for Achieving Fault-Tolerance in On-Chip Networks”. In: *IEEE Transactions on Computers (TC)* 65.2 (Feb. 2016), pp. 353–366. ISSN: 0018-9340. DOI: `10.1109/TC.2015.2425887`.

[111] S. Rodrigo et al. “Cost-Efficient On-Chip Routing Implementations for CMP and MPSoC Systems”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 30.4 (Apr. 2011), pp. 534–547. ISSN: 0278-0070. DOI: `10.1109/TCAD.2011.2119150`.

[112] D. Rossi, P. Angelini, and C. Metra. “Configurable Error Control Scheme for NoC Signal Integrity”. In: *Proc. of 13th IEEE Int’l On-Line Testing Symp. (IOLTS)*. July 2007, pp. 43–48. DOI: `10.1109/IOLTS.2007.24`.

[113] Pradip Kumar Sahu and Santanu Chattopadhyay. “A Survey on Application Mapping Strategies for Network-on-Chip Design”. In: *Journal of Systems Architecture* 59.1 (Jan. 2013), pp. 60–76. ISSN: 1383-7621. DOI: `10.1016/j.sysarc.2012.10.004`.

[114] Jose C. Sancho, Antonio Robles, and Jose Duato. “An Effective Methodology to Improve the Performance of the Up*/Down* Routing Algorithm”. In: *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 15.8 (2004), pp. 740–754. ISSN: 1045-9219. DOI: `10.1109/TPDS.2004.28`.

[115] Prabhat Kumar Saraswat, Paul Pop, and Jan Madsen. “Task Migration for Fault-tolerance in Mixed-criticality Embedded Systems”. In: *Special Issue on the 2nd Int’l Workshop on Adaptive and Reconfigurable Embedded Systems (APRES)* 6.3 (Oct. 2009), 6:1–6:5. ISSN: 1551-3688. DOI: `10.1145/1851340.1851348`.

[116] Takieddine Sbiai and Kazuteru Namba. “NoC Dynamically Reconfigurable as TAM”. In: *Proc. of 21st IEEE Asian Test Symposium (ATS)*. Nov. 2012, pp. 326–331. DOI: `10.1109/ATS.2012.18`.

[117] G. Schley, I. Ahmed, M. Afzal, and M. Radetzki. “Reconfigurable Fault Tolerant Routing for Networks-on-Chip with Logical Hierarchy”. In: *Comput. Electr. Eng.* 51.C (Apr. 2016), pp. 195–206. ISSN: 0045-7906. DOI: `10.1016/j.compeleceng.2016.02.013`.

[118] G. Schley, N. Batzolis, and M. Radetzki. “Fault Localizing End-to-End Flow Control Protocol for Networks-on-Chip”. In: *Proc. of 21st Euromicro Int’l Conf. on Parallel, Distributed and Network-Based Processing (PDP)*. 2013, pp. 454–461. DOI: `10.1109/PDP.2013.74`.

[119] G. Schley and M. Radetzki. “Fault Tolerant Routing for Hierarchically Organized Networks-on-Chip”. In: *Proc. of 23rd Euromicro Int’l Conf. on Parallel, Dis-*

*tributed, and Network-based Processing (PDP)*. Mar. 2015, pp. 379–386. DOI: 10.1109/PDP.2015.36.

[120] G. Schley and M. Radetzki. "Optimal distribution of privileged nodes in networks-on-chip". In: *IEEE Proc. of the Ninth International Workshop on Intelligent Solutions in Embedded Systems (WISES)*. July 2011, pp. 87–92.

[121] G. Schley et al. "Multi-Layer Diagnosis for Fault-Tolerant Networks-on-Chip". In: *IEEE Transactions on Computers (TC)* 66.5 (May 2017), pp. 848–861. ISSN: 0018-9340. DOI: 10.1109/TC.2016.2628058.

[122] T. Schönwald, J. Zimmermann, O. Bringmann, and W. Rosenstiel. "Fully Adaptive Fault-Tolerant Routing Algorithm for Network-on-Chip". In: *Proc of 10th Euromicro Conf. on Digital System Design Architectures, Methods and Tools (DSD)*. Aug. 2007, pp. 527–534. DOI: 10.1109/DSD.2007.4341518.

[123] M.D. Schroeder et al. "Autonet: a high-speed, self-configuring local area network using point-to-point links". In: *IEEE Journal on Selected Areas in Communications* 9.8 (1991), pp. 1318–1335. ISSN: 0733-8716. DOI: 10.1109/49.105178.

[124] Jaume Segura and Charles Hawkins. *CMOS Electronics: How it Works, How it Fails*. 1st. Wiley IEEE Press, 2004. ISBN: 0471476692.

[125] Saeed Shamshiri, Amirali Ghofrani, and Kwang-Ting Cheng. "End-to-end error correction and online diagnosis for on-chip networks". In: *Proc. of IEEE Int'l Test Conf. (ITC)*. 2011, pp. 1–10. DOI: 10.1109/TEST.2011.6139156.

[126] Si2 (Silicon Integration Initiative). visited May 2017. URL: http://www.si2.org/openeda.si2.org/projects/nangatelib.

[127] W. Sirisaengtaksin and S.K. Gupta. "Enhanced crosstalk fault model and methodology to generate tests for arbitrary inter-core interconnect topology". In: *Proc. of the 11th Asian Test Symp. (ATS)*. Nov. 2002, pp. 163–169. DOI: 10.1109/ATS.2002.1181705.

[128] V. Srivastava and M. Motani. "Cross-layer Design: A Survey and the Road Ahead". In: *Communications Magazine* 43.12 (Dec. 2005), pp. 112–119. ISSN: 0163-6804. DOI: 10.1109/MCOM.2005.1561928.

[129] Malgorzata Steinder and Adarshpal S. Sethi. "A survey of fault localization techniques in computer networks". In: *Science of Computer Programming* 53.2 (2004). Topics in System Administration, pp. 165–194. ISSN: 0167-6423. DOI: 10.1016/j.scico.2004.01.010.

[130] Khadija Stewart and Spyros Tragoudas. "Innterconnect Testing for Networks on Chips". In: *Proc. of 24th IEEE VLSI Test Symp. (VTS)*. Apr. 2006, pp. 100–106. DOI: 10.1109/VTS.2006.41.

[131] Tobias Stumpf, Hermann Härtig, Eberle A Rambo, and Rolf Ernst. "Cross-layer Resilience Mechanisms to Protect the Communication Path in Embedded Systems". In: *Proc. of 1st Int'l Workshop on Resiliency in Embedded Electronic Systems (REES)*. Amsterdam, Netherlands, Oct. 2015.

[132] Eduardo Wachter, Augusto Erichsen, Alexandre Amory, and Fernando Moraes. "Topology-agnostic fault-tolerant NoC routing method". In: *Proc. of Design, Automation & Test in Europe Conf. (DATE)*. Mar. 2013, pp. 1595–1600. DOI: 10.7873/DATE.2013.324.

[133] Eduardo Wachter et al. "Runtime fault recovery protocol for NoC-based MPSoCs". In: *Proc. of Fifteenth Int'l Symp. on Quality Electronic Design (ISQED)*. Mar. 2014, pp. 132–139. DOI: 10.1109/ISQED.2014.6783316.

[134] E. Wachter et al. “A layered approach for fault tolerant NoC-based MPSoCs”. In: *Proc. of 17th Latin-American Test Symposium (LATS)*. Apr. 2016, pp. 189–194. DOI: `10.1109/LATW.2016.7483367`.

[135] Laung-Terng Wang, Charles E. Stroud, and Nur A. Touba. *System-on-Chip Test Architectures*. 1st. Morgan Kaufmann, 2008, pp. 351–422. ISBN: 978-0-12-373973-5.

[136] Laung-Terng Wang, Cheng-Wen Wu, and Xiaoqing Wen. *VLSI Test Principles and Architectures: Design for Testability*. 1st. Morgan Kaufmann, 2006. ISBN: 9780123705976.

[137] E. Weldon. “An Improved Selective-Repeat ARQ Strategy”. In: *IEEE Transactions on Communications* 30.3 (Mar. 1982), pp. 480–486. ISSN: 0090-6778. DOI: `10.1109/TCOM.1982.1095497`.

[138] P.M. Wells, K. Chakraborty, and G.S. Sohi. “Adapting to Intermittent Faults in Future Multicore Systems”. In: *Proc. of the 16th Int’l Conf. on Architecture and Compilation Techniques (PACT)*. Sept. 2007, pp. 431–431. DOI: `10.1109/PACT.2007.4336259`.

[139] M.J.Y. Williams and J.B. Angell. “Enhancing Testability of Large-Scale Integrated Circuits via Test Points and Additional Logic”. In: *IEEE Transactions on Computers (TC)* C-22.1 (Jan. 1973), pp. 46–60. ISSN: 0018-9340. DOI: `10.1109/T-C.1973.223600`.

[140] M. Winter, S. Prusseit, and P.F. Gerhard. “Hierarchical routing architectures in clustered 2D-mesh Networks-on-Chip”. In: *Proc. of Int’l SoC Design Conf. (ISOCC)*. 2010, pp. 388–391. DOI: `10.1109/SOCDC.2010.5682890`.

[141] Hans-Joachim Wunderlich and Martin Radetzki. “Multi-Layer Test and Diagnosis for Dependable NoCs”. In: *Proc. of the 9th Int’l Symp. on Networks-on-Chip (NOCS)*. Vancouver, BC, Canada: ACM, 2015, 5:1–5:8. ISBN: 978-1-4503-3396-2. DOI: `10.1145/2786572.2788708`.

[142] Thomas Canhao Xu et al. “Optimal placement of vertical connections in 3D Network-on-Chip”. In: *Journal of Systems Architecture* 59.7 (2013), pp. 441–454.

[143] Heng Yu, Yajun Ha, and B. Veeravalli. “Communication-aware application mapping and scheduling for NoC-based MPSoCs”. In: *Proc. of IEEE Int’l Symp. on Circuits and Systems (ISCAS)*. May 2010, pp. 3232–3235. DOI: `10.1109/ISCAS.2010.5537920`.

[144] Qiaoyan Yu and P. Ampadu. “Adaptive Error Control for NoC Switch-to-Switch Links in a Variable Noise Environment”. In: *Proc. of IEEE Int’l Symp. on Defect and Fault Tolerance of VLSI Systems (DFTVS)*. Oct. 2008, pp. 352–360. DOI: `10.1109/DFT.2008.40`.

[145] C.A. Zeferino, M.E. Kreutz, L. Carro, and A.A. Susin. “A study on communication issues for systems-on-chip”. In: *Proc. of 15th Symp. on Integrated Circuits and Systems Design (SBCCI)*. 2002, pp. 121–126. DOI: `10.1109/SBCCI.2002.1137647`.

[146] Ying Zhang, Huawei Li, and Xiaowei Li. “Reliable Network-on-Chip Router for Crosstalk and Soft Error Tolerance”. In: *Proc. of 17th Asian Test Symp. (ATS)*. Nov. 2008, pp. 438–443. DOI: `10.1109/ATS.2008.35`.

[147] Yixuan Zhang, Randy Morris, Dominic DiTomaso, and Avinash Karanth Kodi. “Energy-Efficient and Fault-Tolerant Unified Buffer and Bufferless Crossbar Architecture for NoCs”. In: *Proc. of 26th IEEE Int’l Parallel and Distributed Pro-*

*cessing Symposium Workshops & PhD Forum, IPDPS*. 2012, pp. 972–981. DOI: 10.1109/IPDPSW.2012.119.

[148] Zhen Zhang et al. “Localization of damaged resources in NoC based shared-memory MP2SOC, using a Distributed Cooperative Configuration Infrastructure”. In: *Proc. of 29th IEEE VLSI Test Symp. (VTS)*. May 2011, pp. 229–234. DOI: 10.1109/VTS.2011.5783726.

# Appendix A
# Appendix

## A.1 Proof of Deadlock Freedom for Hierarchical Routing

This section contains the proof of deadlock freedom for the reconfigurable hierarchical routing presented in Section 5.3. It is divided into two subsections. In Subsection A.1.1 the enumeration order of communication channels is presented that forms the basis of the proof. In Subsection A.1.2, by means of mathematical induction, the proof is given that the routing is deadlock-free for any number of hierarchical levels when using two VCs per level and the allocation scheme described in Subsection 5.3.2.2.

Listing 8 summarizes all the symbols and functions used in the following. Note that for this section some of the symbols are reused and have a different meaning compared to their original definition in the former chapters.

**Listing 8** Summary of Symbols and Functions.

$T$: topology
$V_{h,i}$: set of vertices $v$ representing the subunits of $U_{h,i}$
$C_{h,i}$ set of channels $c$ that connect vertices $v$
$G_{h,i} = (V_{h,i}, C_{h,i})$: graph that represents the abstract topology of unit $U_{h,i}$
$S_{h,i}$: breadth first search spanning tree of $G_{h,i}$
$o$: order of vertices
$a$: enumeration of vertices
$\Sigma$: enumeration of channels
$I$: interval of channel numbers
$p$: path; sequence of channels
$Z$: cycle; cyclic sequence of channels
$dir = \{up, down\}$: direction
$id : V \rightarrow \mathbb{N}$: returns ID $i$ of a vertex $v$
$dist : V \rightarrow \mathbb{N}$: returns the distance of a vertex $v$ to the root vertex of spanning tree $S$
$leaf : V \rightarrow \{true, false\}$: returns true if vertex $v$ is a leaf vertex in spanning tree $S$

### A.1.1 Channel Order

Let graph $G_{h,i}$ be the representation of the abstract topology of unit $U_{h,i}$ with the subunits as vertices $V_{h,i}$ and the interconnection channels between the subunits as edges $C_{h,i}$. Further, let $S_{h,i}$ be the breadth first search spanning tree of $G_{h,i}$. Based on the distance *dist* of vertices $v \in V_{h,i}$ to the root vertex of $S_{h,i}$ an order $o_{h,i}$ can be constructed. According to $o_{h,i}$, a vertex $v_k$ is less than a vertex $v_l$ if the distance of $v_k$ is less than the one of $v_l$. If both vertices have the same distance, $v_k$ is less than $v_l$ either if $v_k$ is a leaf vertex while $v_l$ is not or if both vertices are equivalent and $v_k$ has a lower ID than $v_l$:

$$\begin{aligned}
&\forall v_k, v_l \in V_{h,i} : v_k < v_l \Leftrightarrow \\
&\quad \Big(dist(v_k) < dist(v_l)\Big) \vee \\
&\quad \Big(\Big(dist(v_k) = dist(v_l)\Big) \wedge \\
&\quad \Big(\Big(\neg leaf(v_k) < leaf(v_l)\Big) \vee \\
&\quad \Big(\Big(leaf(v_k) \Leftrightarrow leaf(v_l)\Big) \wedge \Big(id(v_k) < id(v_l)\Big)\Big)\Big)\Big)
\end{aligned} \tag{A.1}$$

Let $a_{h,i} = v_0 v_1 ... v_{y-1}$ be an enumeration of vertices $v \in V_{h,i}$ resulting from order $o_{h,i}$ with $y = |V_{h,i}|$ and $\forall k, l \in [0, y-1], v_k < v_l \Leftrightarrow k < l$. The first element $v_0$ represents the root unit and $v_{y-1}$ represents a leaf unit. In the following, if $v_k < v_l$ then for the corresponding subunits the expression $U_{h-1,k} < U_{h-1,l}$ is used.

To prove that the hierarchical routing is deadlock-free, it is necessary to show that:

- the routing only uses channels with descending channel number and
- when VCs are only allocated in descending order it is not possible that in the *channel dependency graph* (CDG) of $G_{h,i}$ channels can be reached for a second time.

This can be shown by means of an enumeration of channels which assigns a unique number to each channel of every VC.

Let $\Sigma$ be a channel enumeration that assigns a unique number to each channel of every VC. Consequently, an interval $I_\Sigma = \Big[0, \sigma_{max}\Big]$ exists that contains all numbers of $\Sigma$ ranging from 0 to $\sigma_{max}$. As shown in Figure A.1, Interval $I_\Sigma$ is composed of the VC enumeration intervals of each hierarchical level $h$. Starting with the lowest ranked VC $vc_{1,down}$ the VCs are enumerated with in-

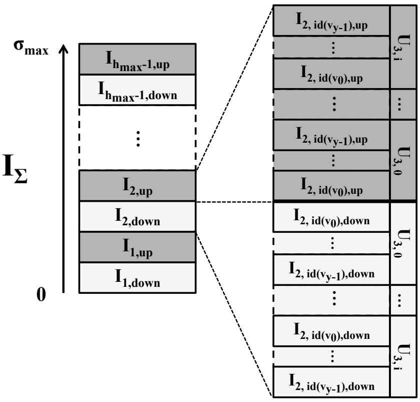


Fig. A.1: Enumeration of channels.

creasing rank up to $vc_{h_{max}-1,up}$. In this way, all channels of a VC $vc_{h,dir}$ have a number within the corresponding interval $I_{h,dir}$. In the following, if an interval $I_i$ contains lower numbers than an interval $I_j$ this is represented by the expression $I_i < I_j$:

$$I_i < I_j \Leftrightarrow \forall k \in I_i \forall l \in I_j : k < l \tag{A.2}$$

Each interval $I_{h,dir}$, in turn, is composed of the intervals $I_{h,i,dir}$ of subunits of every superunit. In case of $vc_{h,down}$, enumeration order of superunits is by decreasing unit ID. For $vc_{h,up}$ it is the other way around. The enumeration order for the subunits of a superunit is defined by $a_{h+1,j}$. In case of a *Down*-VC $vc_{h,down}$ channel enumeration begins at leaf unit $U_{h,id(v_{y-1})}$. After enumeration of all channels of $U_{h,id(v_{y-1})}$, all interconnection channels connecting $U_{h,id(v_{y-1})}$ with other hierarchical units are enumerated. The numbers of these interconnection channels belong to interval $I_{h,id(v_{y-1}),down}$ as well and correspond to the highest numbers in this interval. Subsequently, channel enumeration is continued at unit $U_{h,id(v_{y-2})}$ again followed by enumeration of all its interconnection channels which have not already been enumerated. This enumeration scheme is continued until all channels and interconnection channels of units $U_{h,id(v_k)}$ with $v_k \in a_{h+1,j}$ are enumerated. For *Up*-VC $vc_{h,up}$ channel enumeration is done in a similar way, however, in this case enumeration begins at the root unit $U_{h,id(v_0)}$.

Basically it applies, if according to $a_{h+1,j}$ $v_k < v_l$ then $I_{h,id(v_k),up} < I_{h,id(v_l),up}$ and $I_{h,id(v_k),down} > I_{h,id(v_l),down}$.

### *A.1.2 Proof of Deadlock Freedom*

By means of mathematical induction it is now proven that the reconfigurable hierarchical routing is deadlock-free.

**Proposition:** With the appropriate number of VCs and the used VC allocation scheme, no deadlocks can occur between hierarchical units of same and different levels.

**Base case**
Let $T$ be a network topology with $h_{max} = 2$ hierarchy levels and $|\Theta_1| \geq 2$ hierarchical units on level $h = 1$. Two VCs $vc_{1,up}$ and $vc_{1,down}$ are used, and thus, the resulting enumeration $\Sigma$ consists of two intervals $I_{1,up}$ and $I_{1,down}$.

In such a network, a deadlock situation exists if the corresponding CDG of $T$ contains a cycle $Z = c_0...c_{m-1}c_0$ of length $m$. The channels of cycle $Z$ can either belong to one single hierarchical unit $U_{1,i}$ or to different units. Since within a hierarchical unit $U_{1,i}$ basic *Up/Down* routing is applied which is deadlock-free, no cycle $Z$ can exists that only consists of channels of $U_{1,i}$. For this reason, cycle $Z$ has to span over two or more units of level $h = 1$.

Assume cycle $Z = c_0...c_{m-1}c_0$ spans over the units $U_{1,i}$ and $U_{1,j}$. Without loss of generality $U_{1,i} < U_{1,j}$ according to enumeration $a_{2,0}$, i.e. $U_{1,i}$ is reached in *Up* direction. The channels $c_0...c_{m-1}$ of cycle $Z$ can be subdivided into two paths $p_{ji} = c_0...c_{m-1-x}$ and $p_{ij} = c_{m-x}...c_{m-1}$, i.e. $Z = p_{ji} + p_{ij} + c_0$. In order to form cycle $Z$, channel $c_0$ and $c_{m-1}$ have to be incident to the same node, which is denoted $U_{0,\alpha} \in U_{1,j}$, with $c_0$ originating from $U_{0,\alpha}$ and $c_{m-1}$ leading to $U_{0,\alpha}$. The same applies for channels $c_{m-x-1}$ and $c_{m-x}$, which have to be incident to a node denoted $U_{0,\beta} \in U_{1,i}$.

Since $U_{1,j} > U_{1,i}$, unit $U_{1,j}$ is not the root unit $U_{1,r}$ of level $h = 1$. This implies that at least one path $p_{jr}$ from node $U_{0,\alpha}$ to a node $U_{0,\gamma}$ in root unit $U_{1,r}$ exists that contains only channels $c \in vc_{1,up}$. If unit $U_{0,\beta}$ can be reached by $p_{jr}$, this implies that path $p_{ji}$ is a subpath of $p_{jr}$, and thus, $p_{ji}$ also contains only channels of $vc_{1,up}$. Path $p_{ji}$ must pass through zero or more hierarchical units $U_{1,k}$ using channels of $vc_{1,up}$. According to channel enumeration $\Sigma$ for intervals $I_{1,k,up}$ it applies $I_{1,j,up} > I_{1,k,up} > I_{1,i,up}$.

In case $p_{ji}$ is not a subpath of $p_{jr}$ then there is at least one path $p_{ir}$ that leads from node $U_{0,\beta}$ to node $U_{0,\gamma}$ using only channels of $vc_{1,up}$. There has to be at least one unit $U_{1,l}$ that can be reached by $p_{jr}$ and $p_{ir}$. This for example can be the root unit $U_{1,r}$. Since $U_{1,l}$ can be reached by both paths using channels of $vc_{1,up}$, according to enumeration $a_{2,0}$ unit $U_{1,l}$ has to be less than $U_{1,j}$ and $U_{1,i}$. This implies that a path $p_{jl}$ exists that contains only channels of $vc_{1,up}$ and a path $p_{li}$ that contains only channels of $vc_{1,down}$. Thus, path $p_{ji} = p_{jl} + p_{li}$ contains an *Up* to *Down* turn at unit $U_{1,l}$. The number of the last channel of $p_{jl}$ is within $I_{1,l,up}$ while the number of first channel of $p_{li}$ has to be in interval $I_{1,l,down}$. According to channel enumeration $\Sigma$ $I_{1,j,up} > I_{1,l,up} > I_{1,l,down} > I_{1,i,down}$.

In both cases, $p_{ji}$ being a subpath of $p_{jr}$ or not, path $p_{ji}$ does only consists of channels with descending channel numbers and starts with a channel in *Up* direction. Thus $c_0 \in vc_{1,up}$ and its number cannot be lower than the numbers of $I_{1,l,up}$.

Path $p_{ij}$ leads in opposite direction from $U_{0,\beta}$ to $U_{0,\alpha}$. For this reason, $p_{ij}$ can only consist of channels of $vc_{1,down}$ or may contain an *Up* to *Down* turn like $p_{ji}$. Because $U_{1,j} > U_{1,i}$, however, $p_{ji}$ has to contain at least one channel in *Down* direction and therefore $c_{m-1} \in vc_{1,down}$. The channel number of $c_{m-1}$ has to be within interval $I_{1,j,down}$.

However, since $I_{1,j,down} \in I_{1,down} < I_{1,l,up} \in I_{1,up}$ channel $c_{m-1}$ always has a lower number than $c_0$. Consequently, in the CDG $c_0$ cannot be reached from $c_{m-1}$ since this would contradict the rule that VCs must be allocated in descending order. For this reason, the CDG of $T$ does not contain a cycle $Z$ and thus the routing is deadlock-free for $h_{max} = 2$.

**Inductive step**

Let $\Sigma'$ be an enumeration of channels for $h_{max} - 1$ levels according to the enumeration order described in Subsection A.1.1. The maximum value of $\Sigma'$ is represented by $\sigma'_{max}$. For the additional hierarchy level $h_{max}$, the two VCs are $vc_{h_{max}-1,up}$ and $vc_{h_{max}-1,down}$. Starting from $\sigma'_{max} + 1$, first, the channels of $vc_{h_{max}-1,down}$ and subsequently the channels of $vc_{h_{max}-1,up}$ are enumerated. Thus $I_{h_{max}-1,up} > I_{h_{max}-1,down}$.

For $h_{max} > 2$ more than one hierarchy level exists and thus it has to be ensured that no deadlock between different hierarchy level occurs. Let $Z = c_0...c_{m-1}c_0$ be a cycle of length $m$ in the CDG that spans over different hierarchy levels, i.e. it contains channels of different VCs $vc_{h,dir}$ of different levels $h$. Let $c_0$ and $c_{m-1}$ be incident to node $U_{0,i}$ whereas $c_0$ originates from $U_{0,i}$ while $c_{m-1}$ leads to $U_{0,i}$. According to the VC allocation rule, every channel of $Z$ has to be of the same VC as its predecessor channel or of a lower ranked VC, $\forall i \in [1, m-1] : c_{i-1} \in vc_\alpha \wedge c_i \in vc_\beta$ with $vc_\alpha \geq vc_\beta$. Without loss of generality, let $c_0 \in vc_{h,dir}$ and the channel number of $c_0$ within interval

$I_{h,dir}$. Because cycle $Z$ spans over different hierarchy levels, there has to be at least one channel $c_{m-x}$ that belongs to a VC $vc_{h-1,dir}$ of level $h-1$ with $vc_{h-1,dir} < vc_{h,dir}$. The channel number of $c_{m-x}$ is within $I_{h-1,dir}$ and consequently, $c_{m-x}$ has a lower number than channel $c_0$.

To form cycle $Z$ in the CDG, channel $c_0$ has to be reached from $c_{m-1}$ which requires that the channel number of $c_{m-1}$ is bigger than that of $c_0$. This is only possible if $c_{m-1} \in vc_\alpha$ is of the same VC as $c_0$ or of a higher ranked VC and thus $vc_\alpha \geq vc_{h,dir}$. Because $vc_{h-1,dir} < vc_{h,dir}$ and $vc_\alpha \geq vc_{h,dir}$ this implies that from a lower ranked VC $vc_{h-1,dir}$ a higher ranked VC $vc_\alpha$ has to be allocated. This however, is not possible since it contradicts the VC allocation rule, and thus no cycle $Z$ exist that leads to a deadlock between different hierarchy levels.

Further, no deadlock situation may exist within a level $h$. This case, however, is equivalent to the base case for $h_{max} = 2$. Within a unit $U_{h,i}$ of any level $h$ no cycle $Z$ can form in the CDG because basic *Up/Down* routing is used, which is deadlock-free. In case of a deadlock between hierarchical units of level $h$, a cycle $Z = c_0 ... c_{m-1} c_0$ exists that can be divided into two paths $p_{ji} = c_0 ... c_{m-1-x}$ and $p_{ij} = c_{m-x} ... c_{m-1}$ between two nodes $U_{0,\alpha} \in U_{h,j}$ and $U_{0,\beta} \in U_{h,i}$. The proof that no cycle $Z$ can exist is equivalent to the proof given for the base case for $h_{max} = 2$.

Provided by the VC allocation scheme that VCs may only be allocated by descending rank, the CDG does not contain any cycles $Z$. Consequently, no deadlock situations can occur between hierarchical units of different levels $h$ or between units of the same hierarchy level. For this reason, the reconfigurable hierarchical routing is deadlock-free for any number $h_{max}$ of hierarchy levels. $\blacksquare$

# Index